Description of STM32U5 HAL and low-layer drivers

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

STM32Cube includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per Series (such as STM32CubeU5 for STM32U5 devices)
  - The STM32Cube HAL, STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio. HAL APIs are available for all peripherals.
  - Low-layer APIs (LL) offering a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as Azure® RTOS ThreadX, USBX and ST Network library.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors. The HAL drivers are feature-oriented instead of IP-oriented. For example, the timer APIs are split into several categories following the IP functions, such as basic timer, capture and pulse width modulation (PWM). The HAL driver layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Run-time detection is also suitable for user application development and debugging.

The LL drivers offer hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities, and provide atomic operations that must be called by following the programming model described in the product line reference manual. As a result, the LL services are not based on standalone processes and do not require any additional memory resources to save their states, counter or data pointers. All operations are performed by changing the content of the associated peripheral registers. Unlike the HAL, LL APIs are not provided for peripherals for which optimized access is not a key feature, or for those requiring heavy software configuration and/or a complex upper-level stack (such as USB).

The HAL and LL are complementary and cover a wide range of application requirements:

- The HAL offers high-level and feature-oriented APIs with a high-portability level. These hide the MCU and peripheral complexity from the end-user.
- The LL offers low-level APIs at register level, with better optimization but less portability. These require deep knowledge of the MCU and peripheral specifications.

The HAL- and LL-driver source code is developed in Strict ANSI-C, which makes it independent of the development tools. It is checked with the Coverity static analysis tool. It is fully documented.

It is compliant with MISRA C®:2012 standard.

This user manual is structured as follows:
- Overview of HAL drivers
- Overview of low-layer drivers
- Cohabiting of HAL and LL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application

1 General information

The STM32CubeU5 MCU Package runs on STM32U5 32-bit microcontrollers based on the Arm® Cortex®-M processor with Arm® TrustZone® technology.

Note: Arm and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Acronyms and definitions

Table 1. **Acronyms and definitions**

Acronym	Definition
ADC	Analog-to-digital converter
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CAN	Controller area network
CMSIS	Cortex microcontroller software interface standard
COMP	Comparator
CORDIC	Trigonometric calculation unit
CPU	Central processing unit
CRC	CRC calculation unit
CRYP	Cryptographic processor
CSS	Clock security system
DAC	Digital to analog converter
DLYB	Delay block
DCMI	Digital camera interface
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
DSI	Display serial interface
EXTI	External interrupt/event controller
FDCAN	Flexible data-rate controller area network unit
FLASH	Flash memory
FMAC	Filtering mathematical calculation unit
FMC	Flexible memory controller
GFXMMU	Graphical oriented memory management unit (Chrom-GRC)
GFXTIM	Graphic timer
GPIO	General-purpose I/Os
GPU2D	Graphic processing unit (Neo-Chrom)
GTZC	Global TrustZone® controller
GTZC-MPCBB	GTZC block-based memory protection controller
GTZC-MPCWM	GTZC watermark memory protection controller
GTZC-TZIC	TrustZone® illegal access controller
GTZC-TZSC	TrustZone® security controller
HAL	Hardware abstraction layer
HASH	Hash processor
HCD	USB host controller driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound

Acronym	Definition
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog
JPEG	Joint photographic experts group
LCD	Liquid crystal display controller
LTDC	Liquid crystal display - thin film transistor display controller
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MDIOS	Management data input/output (MDIO) slave
MDMA	Master direct memory access
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND Flash memory
NOR	NOR Flash memory
NVIC	Nested vectored interrupt controller
OCTOSPI	Octo-SPI interface
OPAMP	Operational amplifier
OTFDEC	On-the-fly decryption engine
PKA	Public key accelerator
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PSSI	Parallel synchronous slave interface
PWR	Power controller
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SAI	Serial audio interface
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial peripheral interface
SRAM	SRAM external memory
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch sensing controller
UART	Universal asynchronous receiver/transmitter
UCPD	USB Type-C® and Power Delivery interface
USART	Universal synchronous receiver/transmitter

Acronym	Definition
USB	Universal serial bus
VREFBUF	Voltage reference buffer
WWDG	Window watchdog

3 Overview of HAL drivers

The HAL drivers are designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names.

The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode
- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (such as USART1 or USART2)
- All HAL APIs implement user-callback functions mechanism:
  - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
  - Peripherals interrupt events
  - Error events
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

3.1 HAL and user-application files

3.1.1 HAL driver files

HAL drivers are composed of the following set of files:

Table 2. HAL driver files

File	Description
stm32u5xx_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32u5xx_hal_adc.c, stm32u5xx_hal_irda.c.*
stm32u5xx_hal_ppp.h	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. *Example:stm32u5xx_hal_adc.h,stm32u5xx_hal_irda.h.*
stm32u5xx_hal_ppp_ex.c	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. *Example:stm32u5xx_hal_adc_ex.c,stm32u5xx_hal_flash_ex.c.*
stm32u5xx_hal_ppp_ex.h	Header file of the extension C file It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs *Example: stm32u5xx_hal_adc_ex.h,stm32u5xx_hal_flash_ex.h.*
stm32u5xx_hal.c	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on SysTick APIs.
stm32u5xx_hal.h	stm32u5xx_hal.c header file
stm32u5xx_hal_msp_template.c	Template file to be copied to the user application folder It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32u5xx_hal_conf_template.h	Template file allowing to customize the drivers for a given application
stm32u5xx_hal_def.h	Common HAL resources such as common define statements, enumerations, structures and macros

3.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3. User-application files

File	Description
system_stm32u5xx.c	When TrustZone® security is disabled, this file contains SystemInit() that is called at startup just after reset and before branching to the main program. It allows relocating the vector table in internal Flash memory or internal SRAM.
system_stm32u5xx_ns.c	When TrustZone® security is enabled, this file is part of the non-secure application. It contains SystemInit() that is called by the secure application before branching to the non-secure main program. It allows relocating the vector table in internal Flash memory or internal SRAM.
startup_stm32u5xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.

File	Description
stm32u5xx_flash.icf/ stm32u5xx_flash.icf	When TrustZone® security is disabled, this file is the linker file for EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the application requirements.
stm32u5xx_flash_s.icf/ stm32u5xx_flash_s.icf	When TrustZone® security is enabled, this file is the linker file for EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the secure application requirements.
stm32u5xx_flash_ns.icf/ stm32u5xx_flash_ns.icf	When TrustZone® security is enabled, this file is the linker file for EWARM toolchain. Its main function is the adaptation of the stack/heap size to fit the non-secure application requirements.
stm32u5xx_RAMxspi1_ROMxspi2.icf	This file is the linker file for EWARM toolchain. Its main function is to describe the memory mapping (internal and external memories) available on the STM32H7S78-DK board.
stm32u5xx_hal_msp.c	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
stm32u5xx_hal_conf.h	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32u5xx_it.c/.h	This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in *stm32u5xx*_hal.c) used as HAL timebase. By default, this function is called each 1ms in SysTick ISR. The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.
main.c/.h	This file contains the main program routine, mainly: • Call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

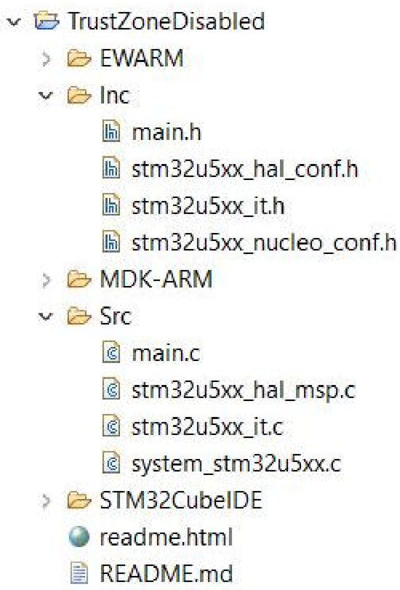
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Its features are the following:

- It contains the sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows configuring the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
  - HAL is initialized
  - SysTick ISR implemented for HAL_GetTick()
  - System clock configured with the selected device frequency.

Note: If an existing project is copied to another location, then include paths must be updated.

Figure 1. Example of project template



3.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

3.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that enables working with several IP instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi-instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
  Example: global pointers, DMA handles, state machine.
- Storage : this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
USART_TypeDef *Instance; /* USART registers base address */
USART_InitTypeDef Init; /* Usart communication parameters */
uint8_t *pTxBuffPtr;/* Pointer to Usart Tx transfer Buffer */
uint16_t TxXferSize; /* Usart Tx Transfer size */
__IO uint16_t TxXferCount;/* Usart Tx Transfer Counter */
uint8_t *pRxBuffPtr;/* Pointer to Usart Rx transfer Buffer */
uint16_t RxXferSize; /* Usart Rx Transfer size */
__IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
HAL_LockTypeDef Lock; /* Locking object */
__IO HAL_USART_StateTypeDef State; /* Usart communication state */
```

```
__IO HAL_USART_ErrorTypeDef ErrorCode;/* USART Error code */
}USART_HandleTypeDef;
```

Note:

1. *The multi-instance feature implies that all the APIs used in the application are reentrant and avoid using global variables because subroutines can fail to be reentrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:*
   – *Reentrant code does not hold any static (or global) non-constant data: reentrant functions can work with global data. For example, a reentrant interrupt service routine can grab a piece of hardware status to work with (for example serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.*
   – *Reentrant code does not modify its own code.*
2. *When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.*
3. *For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:*
   – *GPIO*
   – *SYSTICK*
   – *NVIC*
   – *PWR*
   – *RCC*
   – *FLASH*

3.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
{
  uint32_t BaudRate;       /*!< This member configures the Usart communication baud rate.
                                The baud rate is computed using the following formula:
                                Baud Rate Register[15:4] = ((2 * fclk_pres) /
                                ((huart->Init.BaudRate)))[15:4]
                                Baud Rate Register[3]    = 0
                                Baud Rate Register[2:0]  =  (((2 * fclk_pres) /
                                ((huart->Init.BaudRate)))[3:0]) >> 1
                                where fclk_pres is the USART input clock frequency (fclk)
                                divided by a prescaler.
                                @note  Oversampling by 8 is systematically applied to
                                achieve high baud rates. */
  uint32_t WordLength;          /*!< Specifies the number of data bits
                                transmitted or received in a frame.This
              parameter can be a value of                                               USAR
TEx_Word_Length. */
  uint32_t StopBits;            /*!< Specifies the number of stop bits transmitted.
                                This parameter can be a value of USART_Stop_Bits. */
  uint32_t Parity;              /*!< Specifies the parity mode.
                                This parameter can be a value of USART_Parity
                                @note When parity is enabled, the computed parity is inserted
                                at the MSB position of the transmitted data (9th bit when
                                the word length is set to 9 data bits; 8th bit when the
                                word length is set to 8 data bits). */
  uint32_t Mode;                /*!< Specifies whether the Receive or
              Transmit mode is enabled or disabled.
                                This parameter can be a value of
              USART_Mode. */
  uint32_t CLKPolarity;         /*!< Specifies the steady state of the serial clock.
                                This parameter can be a value of USART_Clock_Polarity. */
  uint32_t CLKPhase;            /*!< Specifies the clock transition on
              which the bit capture is made.
                                This parameter can be a value of USART_Clock_Phase. */
  uint32_t CLKLastBit;          /*!< Specifies whether the clock pulse
              corresponding to the last transmitted data                              bit
(MSB) has to be output on the SCLK pin                              in synchronous mode.
                                This parameter can be a value of USART_Last_Bit. */
  uint32_t ClockPrescaler;      /*!< Specifies the prescaler value used to
              divide the USART clock source.
                                This parameter can be a value of USART_ClockPrescaler. */
} USART_InitTypeDef;
```

Note: The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)
```

3.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc,PPP_ProcessConfig* sConfig)
```

3.3 API classification

The HAL APIs are classified into the following categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL driver files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:**
  This set of API is divided into two sub-categories :
  - **Family specific APIs**: APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t Si
ngleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDi
ff);
```

  - **Device part number specific APIs:**These APIs are implemented in the extension file and delimited by specific define statements relative to a given part number.

Note: The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4. API classification

	Generic file	Extension file
Common APIs	X	X
Family specific APIs	-	X
Device specific APIs	-	X

Note: Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.

The IRQ handlers are used for common and family specific processes.

3.4 Devices supported by HAL/LL drivers

Table 5. List of devices supported by HAL/LL drivers

Name	STM32U575xx	STM32U585xx	STM32U535xx	STM32U545xx	STM32U595xx	STM32U599xx	STM32U5A5xx	STM32U5A9xx	STM32U5G9xx	STM32U5F9xx	STM32U5G7xx	STM32U5F7xx
stm32u5xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_cordic.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_cryp.c	No	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	No
stm32u5xx_hal_cryp_ex.c	No	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	No
stm32u5xx_hal_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dac_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dcache.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dcmi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dma2d.c	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_dsi.c	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No	No
stm32u5xx_hal_exti.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_fdcan.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_fmac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_gfxmmu.c	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_gfxtim.c	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes
stm32u5xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_gpu2d.c	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
stm32uSxx_hal_gtzc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_hash.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_hash_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_hcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_icache.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Name	STM32U575xx	STM32U585xx	STM32U535xx	STM32U545xx	STM32U595xx	STM32U599xx	STM32U5A5xx	STM32U5A9xx	STM32U5G9xx	STM32U5F9xx	STM32U5G7xx	STM32U5F7xx
stm32u5xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_jpeg.c	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes
stm32u5xx_hal_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_ltdc.c	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_ltdc_ex.c	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_mdf.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_mmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_mmc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_msp_template.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_nand.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_nor.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_opamp_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_ospi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_otfdec.c	No	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	No
stm32u5xx_hal_pcd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_pcd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_pka.c	No	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	No
stm32u5xx_hal_pssi.c	No	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_ramcfg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_rng_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_sai.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_sai_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_sd.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_sd_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_smbus_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Name	STM32U575xx	STM32U585xx	STM32U535xx	STM32U545xx	STM32U595xx	STM32U599xx	STM32U5A5xx	STM32U5A9xx	STM32U5G9xx	STM32U5F9xx	STM32U5G7xx	STM32U5F7xx
stm32u5xx_hal_sram.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_timebase_rtc_alarm_template.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_timebase_rtc_wakeup_template.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_timebase_tim_template.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_tsc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_usart_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_hal_xspi.c	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_adc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_comp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_cordic.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_crc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_crs.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_dac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_dlyb.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_dma.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_dma2d.c	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_exti.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_fmc.c	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_fmac.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_gpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_i2c.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_icache.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_lpgpio.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_lptim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_lpuart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_opamp.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_pka.c	No	Yes	No	Yes	No	No	Yes	Yes	Yes	No	Yes	No
stm32u5xx_ll_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_rcc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_rng.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_rtc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_sdmmc.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Name	STM32U575xx	STM32U585xx	STM32U535xx	STM32U545xx	STM32U595xx	STM32U599xx	STM32U5A5xx	STM32U5A9xx	STM32U5G9xx	STM32U5F9xx	STM32U5G7xx	STM32U5F7xx
stm32u5xx_ll_spi.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_tim.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_ucpd.c	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_usart.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_usb.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32u5xx_ll_utils.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

3.5 HAL driver rules

3.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6. **HAL API naming rules**

	Generic	Family specific APIs	Device specific APIs
File names	*stm32u5xx_hal_ppp (c/h)*	*stm32u5xx_hal_ppp_ex (c/h)*	*stm32u5xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function* *HAL_PPP_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*	*HAL_PPPEx_Function* *HAL_PPPEx_FeatureFunction_MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with _TypeDef.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32U5 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in the stm32u5xxx.h header file:
  stm32u5xxx.h corresponds to stm32u535xx.h, stm32u545xx, stm32u575xx.h, stm32u585xx,stm32u595xx.h, stm32u5A5xx.h, stm32u599xx.h and stm32u5A9xx.h.
- Peripheral function names are prefixed by HAL_, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (for example HAL_UART_Transmit()). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named PPP_InitTypeDef (for example ADC_InitTypeDef).
- The structure containing the Specific configuration parameters for the PPP peripheral are named PPP_xxxxConfTypeDef (for example ADC_ChannelConfTypeDef).
- Peripheral handle structures are named PPP_HandleTypedef (e.g DMA_HandleTypeDef)
- The functions used to initialize the PPP peripheral according to parameters specified in PPP_InitTypeDef are named HAL_PPP_Init (for example HAL_TIM_Init()).
- The functions used to reset the PPP peripheral registers to their default values are named HAL_PPP_DeInit (for example HAL_TIM_DeInit()).
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: *HAL_PPP_Function_DMA()*.
- The **Feature** prefix should refer to the new feature.
  Example: *HAL_ADCEx_InjectedStart()*() refers to the injection mode.

3.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
  - GPIO
  - SYSTICK
  - NVIC
  - RCC
  - FLASH

Example: The *HAL_GPIO_Init()* requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
/*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below:

Note: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7. Macros handling interrupts and specific clock configurations

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status
__HAL_PPP_ENABLE(__HANDLE__)	Enables a peripheral
__HAL_PPP_DISABLE(__HANDLE__)	Disables a peripheral
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)	Specific PPP HAL driver macro
__HAL_PPP_GET_ IT_SOURCE (__HANDLE__, __ INTERRUPT __)	Checks the source of specified interrupt

- NVIC and SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the stm32u5xx_hal_cortex.c file.
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
 HAL_PPP_Init(PPP_HandleTypeDef)
if(hppp == NULL)
 {
 return HAL_ERROR;
 }
```

- The macros defined below are used:
  - Conditional macro:

```
#define ABS(x) (((x) > 0) ? (x) : -(x))
```

  - Pseudo-code macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
(__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
(__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32u5xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8. **Callback functions**

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Example: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Example: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Example: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

3.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **IO operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and Errors functions**: HAL_PPP_GetState (), HAL_PPP_GetError ().

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (such as PWM, OC and IC).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The *HAL_DeInit()* function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in run time the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 9. HAL generic APIs

Function group	Common API name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in run time the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in run time the error that occurred during IT routine

3.7 HAL extension APIs

3.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, stm32u5xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32u5xx_hal_ppp_ex.h) for a given part number.

Below an example based on the ADC peripheral:

Table 10. HAL extension APIs

Function group	Common API name
HAL_ADCEx_CalibrationStart()	This function is used to start the automatic ADC calibration
HAL_ADCEx_InjectedStart	This function is used to start the conversion of injected groups.

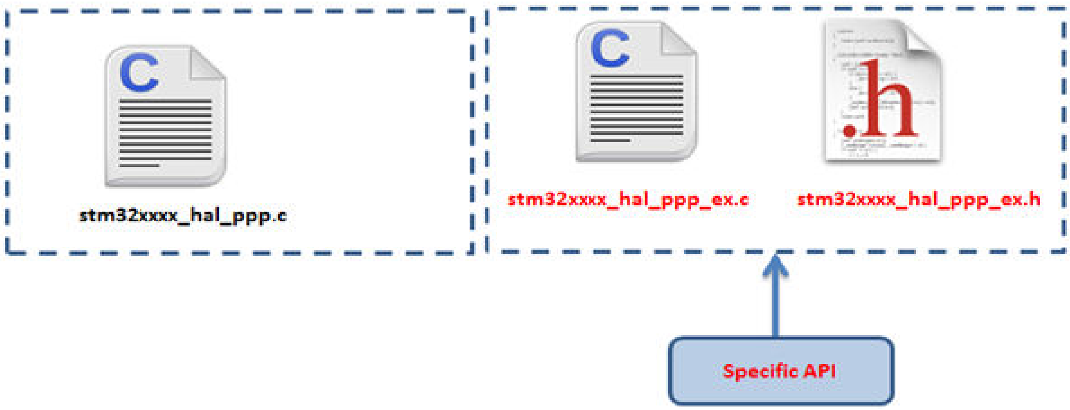
3.7.2 HAL extension model cases

The specific peripheral features can be handled by the HAL drivers in five different ways. They are described below.

Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the stm32u5xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

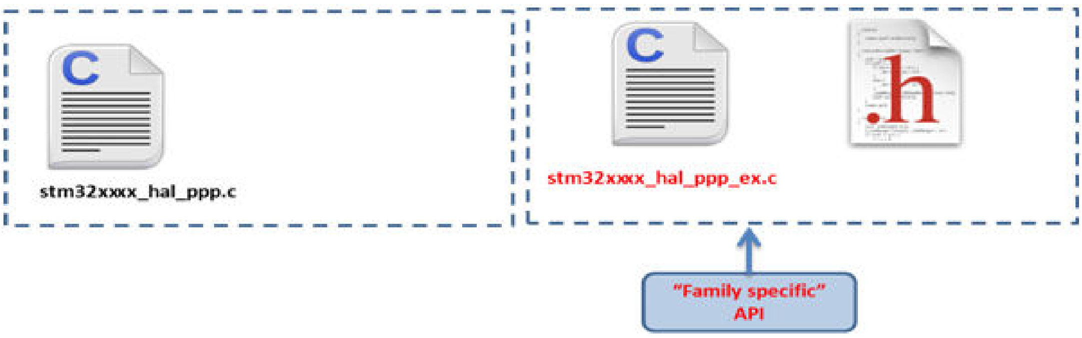
Figure 2. Adding device-specific functions



Adding a family-specific function

In this case, the API is added in the extension driver C file and named HAL_PPPEx_Function ().

Figure 3. Adding family-specific functions

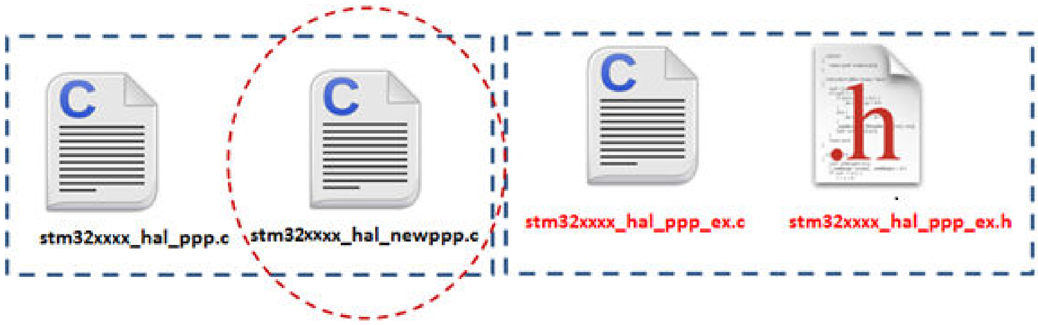


Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32u5xx_hal_newppp.c. However the inclusion of this file is selected in the stm32u5xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. Adding new peripherals

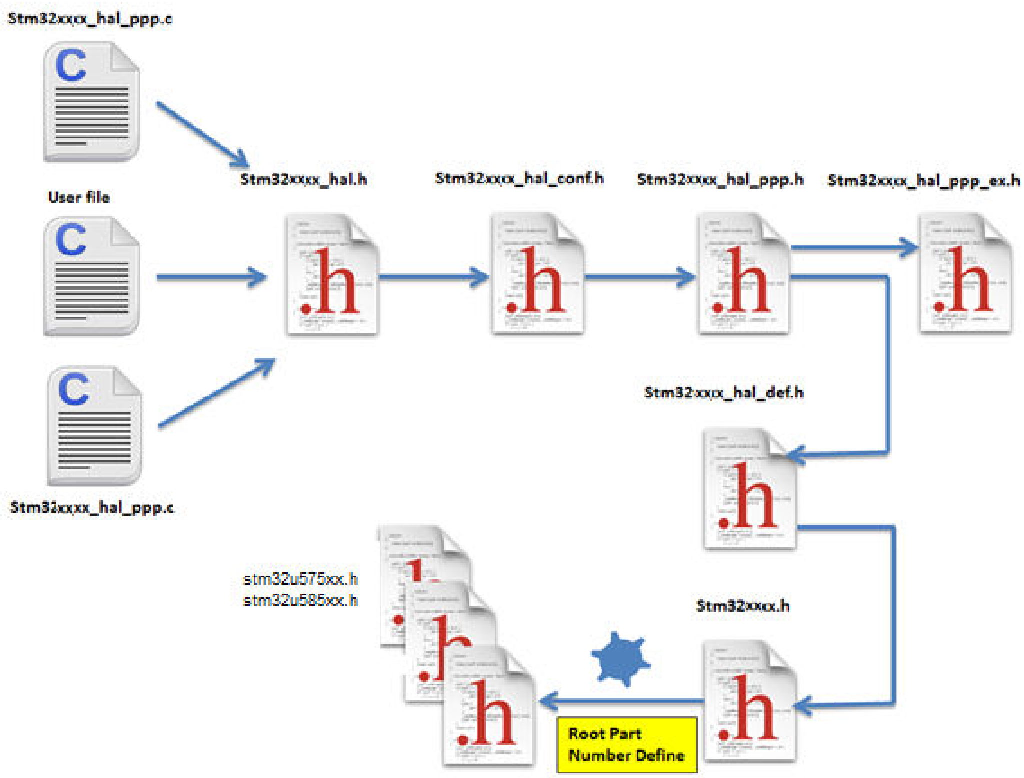


Example: stm32u5xx_hal_adc.c/h

3.8 File inclusion model

The header of the common HAL driver file (stm32u5xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 5. File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```
/*********************************************************************
* @file stm32u5xx_hal_conf.h
* @author MCD Application Team
* @brief   HAL configuration file.
*********************************************************************
(…)
#define HAL_USART_MODULE_ENABLED
#define HAL_IRDA_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
(…)
```

3.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm32u5xx_hal_def.h.*The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status**
  The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
Typedef enum
{
HAL_OK = 0x00,
HAL_ERROR = 0x01,
HAL_BUSY = 0x02,
HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked**
  The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;
```

  In addition to common resources, the stm32u5xx_hal_def.h file calls the stm32u5xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:
  – Declarations of peripheral registers and bits definition.
  – Macros to access peripheral registers hardware (such as Write register or Read register).
- **Common macros**
  – Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

  – Macro linking a PPP peripheral to a DMA structure pointer:

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
 (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
 (__DMA_HANDLE_).Parent = (__HANDLE__); \
} while(0)
```

3.10 HAL configuration

The configuration file, stm32u5xx_hal_conf.h, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable or modify some options by uncommenting, commenting or modifying the values of the related define statements as described in the table below:

Table 11. Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	16 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE start-up, expressed in ms	100 ms
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 Hz
HSI48_VALUE	Value of the internal high speed oscillator for USB FS/SDMMC/RNG expressed in Hz. The real value may vary depending on manufacturing process.	48000000 Hz
MSI_VALUE	Defines the default value of the Multiplespeed internal oscillator (MSI) expressed in Hz.	4 000 000 Hz

Configuration item	Description	Default Value
LSI_VALUE	Defines the default value of the Low-speed internal oscillator (LSI) expressed in Hz.	32000 Hz
LSI_STARTUP_TIMEOUT	Timeout for LSI startup, expressed in ms	130 ms
LSE_VALUE	Defines the value of the external oscillator (LSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 Hz
LSE_STARTUP_TIMEOUT	Timeout for LSE start-up, expressed in ms	5000 Hz
VDD_VALUE	V_{DD} value	3300 mV
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch	TRUE

Note: *The stm32u5xx_hal_conf_template.h file is located in the HAL drivers Inc folder. It should be copied to the user folder, renamed and modified as described above.*

By default, the values defined in the stm32u5xx_hal_conf_template.h file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

3.11 HAL system peripheral handling

This section gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

3.11.1 Clocks

Two main functions can be used to configure the system clock:

- HAL_RCC_OscConfig (RCC_OscInitTypeDef *RCC_OscInitStruct). This function configures/enables multiple clock sources (HSE, HSI, HSI48, MSI, LSE, LSI, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - selects the system clock source
  - configures AHB, APB1, and APB2 clock dividers
  - configures the number of Flash memory wait states
  - updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC, USB). In this case, the clock configuration is performed by an extended API defined in stm32u5xx_hal_rcc_ex.c: *HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() Clock de-initialization function that returns clock configuration to reset state
- Get clock functions that allow retreiving various clock configurations (such as system clock, HCLK, PCLK1 or PCLK2)
- MCO and CSS configuration functions

A set of macros are defined in stm32u5xx_hal_rcc.h and stm32u5xx_hal_rcc_ex.h. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- __HAL_PPP_CLK_ENABLE/__HAL_PPP_CLK_DISABLE to enable/disable the peripheral clock.
- __HAL_PPP_FORCE_RESET/__HAL_PPP_RELEASE_RESET to force/release peripheral reset
- __HAL_PPP_CLK_SLEEP_ENABLE/__HAL_PPP_CLK_SLEEP_DISABLE to enable/disable the peripheral clock during Sleep mode.
- __HAL_PPP_IS_CLK_ENABLED/__HAL_PPP_IS_CLK_DISABLED to query about the enabled/disabled status of the peripheral clock.
- __HAL_PPP_IS_CLK_SLEEP_ENABLED/__HAL_PPP_IS_CLK_SLEEP_DISABLED to query about the enabled/disabled status of the peripheral clock during Sleep mode.

3.11.2 GPIOs

GPIO HAL APIs are the following:

- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
- HAL_GPIO_TogglePin ().
- HAL_GPIO_LockPin()
- HAL_GPIO_EnableHighSPeedLowVoltage()/HAL_GPIO_DisableHighSPeedLowVoltage()

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32u5xx_it.c and implement HAL_GPIO_EXTI_Callback().

The table below describes the GPIO_InitTypeDef structure field.

Table 12. Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: • GPIO mode – GPIO_MODE_INPUT : Input floating – GPIO_MODE_OUTPUT_PP : Output push-pull – GPIO_MODE_OUTPUT_OD : Output open drain – GPIO_MODE_AF_PP : Alternate function push-pull – GPIO_MODE_AF_OD : Alternate function open drain – GPIO_MODE_ANALOG : Analog mode – GPIO_MODE_ANALOG_ADC_CONTROL: ADC analog mode • External Interrupt mode – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • External Event mode – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_FREQ_LOW GPIO_SPEED_FREQ_MEDIUM GPIO_SPEED_FREQ_HIGH GPIO_SPEED_FREQ_VERY_HIGH

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs:

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

3.11.3 Cortex® NVIC and SysTick timer

The Cortex® HAL driver, stm32u5xx_hal_cortex.c, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL_NVIC_SetPriority()/ HAL_NVIC_SetPriorityGrouping()
- HAL_NVIC_GetPriority() / HAL_NVIC_GetPriorityGrouping()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_NVIC_GetActive(IRQn)
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()
- HAL_MPU_Enable()
- HAL_MPU_Enable_NS()
- HAL_MPU_Disable()
- HAL_MPU_Disable_NS()
- HAL_MPU_ConfigRegion()
- HAL_MPU_ConfigRegion_NS()
- HAL_MPU_ConfigMemoryAttributes()
- HAL_MPU_ConfigMemoryAttributes_NS()
- MPU_ConfigRegion()
- MPU_ConfigMemoryAttributes()

3.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
  - HAL_PWR_PVD_IRQHandler()
  - HAL_PWR_PVDCallback()
- Wakeup pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()

- Low-power mode entry
  – HAL_PWR_EnterSLEEPMode()
  – HAL_PWR_EnterSTOPMode() (kept for compatibility with other families)
  – HAL_PWR_EnterSTANDBYMode()

  STM32U5 low-power management features:
  – HAL_PWREx_EnterSTOP0Mode()
  – HAL_PWREx_EnterSTOP1Mode()
  – HAL_PWREx_EnterSTOP2Mode()
  – HAL_PWREx_EnterSHUTDOWNMode()
  – HAL_PWR_EnableSleepOnExit() / HAL_PWR_DisableSleepOnExit: (low-power management Sleep-on-entry)
  – HAL_PWR_EnableSEVOnPend() / HAL_PWR_DisableSEVOnPend (low-power management Cortex-SEV-on-pend)

3.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral, that are handled through EXTI HAL APIs. In addition, each peripheral HAL driver implements the associated EXTI configuration and function as macros in its header file.

The first 16 EXTI lines connected to the GPIOs, are managed within the GPIO driver. The GPIO_InitTypeDef structure allows configuring an I/O as external interrupt or external event.

The EXTI lines connected internally to the PVD, RTC, USB, and Ethernet, are configured within the HAL drivers of these peripheral through the macros given in the table below.

The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13. **Description of EXTI configuration macros**

Macros	Description
__HAL_PPP_{SUBLOCK}__EXTI_ENABLE_IT()	Enables a given EXTI line interrupt Example: __HAL_PWR_PVD_EXTI_ENABLE_IT()
__HAL_PPP_{SUBLOCK}__EXTI_DISABLE_IT()	Disables a given EXTI line. Example: __HAL_PWR_PVD_EXTI_DISABLE_IT()
__HAL_ PPP_{SUBLOCK}__EXTI_GET_FLAG()	Gets a given EXTI line interrupt flag pending bit status. Example: __HAL_PWR_PVD_EXTI_GET_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_CLEAR_FLAG()	Clears a given EXTI line interrupt flag pending bit. Example; __HAL_PWR_PVD_EXTI_CLEAR_FLAG()
__HAL_ PPP_{SUBLOCK}_EXTI_GENERATE_SWIT()	Generates a software interrupt for a given EXTI line. Example: __HAL_PWR_PVD_EXTI_ GENERATE_SWIT ()
__HAL_PPP_SUBBLOCK_EXTI_ENABLE_EVENT()	Enable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_ENABLE_EVENT()
__HAL_PPP_SUBBLOCK_EXTI_DISABLE_EVENT()	Disable a given EXTI line event Example: __HAL_RTC_WAKEUP_EXTI_DISABLE_EVENT()

Macros	Description
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_EDGE()	Configure an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_EDGE()	Disable an EXTI Interrupt or Event on rising edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_ENABLE_RISING_FALLING_EDGE()	Enable an EXTI Interrupt or Event on Rising/Falling edge
__HAL_ PPP_SUBBLOCK_EXTI_DISABLE_RISING_FALLING_EDGE()	Disable an EXTI Interrupt or Event on Rising/Falling edge

If the EXTI interrupt mode is selected, the user application must call HAL_PPP_FUNCTION_IRQHandler() (for example HAL_PWR_PVD_IRQHandler()), from stm32u5xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

3.11.6 DMA

DMA transfer modes are divided into two main categories:

- Normal transfers (as for legacy Series)
- Linked-list transfers

In addition to linked-list capability, all the advanced DMA features are managed and configured via the extended API as extensions to Normal mode. The advanced features are repeated block, trigger and data handling.

For a given channel, enable and configure the peripheral to be connected to the DMA channel (except for internal SRAM and Flash memories for which no initialization is required). Refer to the device reference manuals for details on the connection between peripherals and DMA requests.

The Normal transfer mode is initialized via the standard API while the Linked-list mode is configured via the extended API.

Alternatively to the Normal programming mode, a DMA channel can be programmed by a list of transfers, also known as linked-list (list of node items), each node being defined by its data structure and specifying a standalone DMA channel. When enabled, the DMA channel fetches the first linked-list node from SRAM (also known as head node). When the execution is complete, it fetches and executes the next linked-list node. This operation is repeated until the end of the linked-list queue. The linked-list can be linear or circular:

- Linear linked-list:
  The last linked-list queue node is not linked to another queue node. The DMA channel fetches and executes the whole DMA linked-list queue from the first node (head node) to the last (tail node). When the last node execution is complete, the DMA channel remains in idle state and another transfer can be launched.
- Circular linked-list:
  The last linked-list node can be linked to any linked-list queue node. The DMA channel fetches and executes the whole DMA linked-list queue from the first node to the last. When the last node is executed, the DMA channel fetches again the first circular node and repeats the same sequence in an infinite loop (circular transfer). An abort operation is required to stop the DMA channel. The Circular linked-list mode replaces the legacy Circular transfer mode.

To reduce linked-list queue execution time and power consumption, the DMA channel supports dynamic linked-list execution in two types of formats, static and dynamic:

- Static linked-list format: this format refers to the full linked-list node where all DMA channel parameters are fetched and executed independently of the redundancy of information.
- Dynamic linked-list: this format refers to the customized linked-list node where only the necessary DMA channel parameters are fetched and executed (for example, no update is required if data size = 20 on previous node, and data size = 20 on the current node).

For linked-list transfers, the DMA channel can execute the linked-list queue node by node. This feature is named link-step mode. When activated, enabling the DMA channel for the first time fetches the head node from memory and stops. Another DMA channel enable is then required to execute the node. After that, the DMA channel must remain enabled to execute each node until the end of the linked-list queue. When the linked-list queue is circular, the DMA channel must be enabled in an infinite loop to keep the DMA channel running. This feature is useful for debug purposes or for executing queue nodes asynchronously.

Each DMA channel transfer (normal or linked-list) can be configured depending on the DMA channel instance embedded in the devices. The configurations are the following:

- Repeated block configuration:
  This feature is also named two-dimension addressing transfer. If it is supported, the DMA channel can perform repeated transfers of programmed blocks (the size of the block transfer is the legacy data size). In addition to the block repeat count, the DMA channel addresses can jump at burst or block level. The jump length is a programmable parameter defined in the user application:
  – Jump at burst level: the DMA channel keeps an empty area between two consecutive transmitted bursts.
  – Jump at block level; the DMA channel keeps an empty area between two consecutive transmitted blocks .
- Trigger configuration:
  The DMA channel transfers can be triggered on the rising or falling edge of a given signal (hardware triggers). The trigger condition can be applied at single/burst level, block level, repeated block level or node level.
  The DMA channel can report a trigger overrun when more than two trigger signal events are detected before executing the current transfer.
- Data handling:
  The data handling feature is based on a FIFO capability that can be:
  – Padding pattern: the padding pattern (zero padding or sign extension) is selected when the source data width is smaller than the destination data width.
  – Truncation: the source data is truncated when the source data width is bigger than the destination data width.
  – Pack/unpack: a set of data can be packed when the source data width is smaller than the destination data width or unpacked when the source data width is bigger than the destination data width.
  – Exchange: data can be exchanged at byte and half-word level on the destination, and byte level on the source.

Each ongoing DMA channel transfer (normal or linked-list) can be suspended and resumed at run time by the application. When trying to suspend an ongoing transfer, the DMA channel is not suspended instantly. Instead, the ongoing single/burst transfer completes and then stops.

When the DMA channel is suspended, the current transfer can be resumed instantly.

The DMA channel that supports the FIFO can report in real time the number of beats remaining on the destination FIFO ((output).

Two operating modes are available:

- Polling mode I/O operation
  1. Use HAL_DMA_Start() to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
  2. Use HAL_DMA_PollForTransfer() to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
  1. Configure the DMA interrupt priority using HAL_NVIC_SetPriority().
  2. Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ().
  3. Use HAL_DMA_Start_IT() to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
  4. Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine.
  5. When data transfer is complete, HAL_DMA_IRQHandler() function is executed and a user function can be called by customizing XferCpltCallback and XferErrorCallback function pointer (that is a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer.

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enables the specified DMA channel

- __HAL_DMA_DISABLE: disables the specified DMA channel
- __HAL_DMA_GET_FLAG: gets the DMA channel pending flags
- __HAL_DMA_CLEAR_FLAG: clears the DMA channel pending flags
- __HAL_DMA_ENABLE_IT: enables the specified DMA channel interrupts
- __HAL_DMA_DISABLE_IT: disables the specified DMA channel interrupts
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA channel interrupt has been enabled or not

To correctly use the DMA linked-list feature, apply the following call sequences:

- Linear transfer:
  - Linked-list queue building
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_ConvertQToDynamic()
  - Linked-list queue execution
    - HAL_DMAEx_List_Init()
    - HAL_DMAEx_List_LinkQ()
    - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
    - HAL_DMAEx_List_UnLinkQ()
    - HAL_DMAEx_List_DeInit()
- Circular transfer:
  - Linked-list queue building
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_BuildNode()
    - HAL_DMAEx_List_InsertNode_Tail()
    - HAL_DMAEx_List_SetCircularModeConfig()/HAL_DMAEx_List_SetCircularMode()
    - HAL_DMAEx_List_ConvertQToDynamic()
  - Linked-list queue execution
    - HAL_DMAEx_List_Init()
    - HAL_DMAEx_List_LinkQ()
    - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
    - HAL_DMA_Abort() / HAL_DMA_Abort_IT()
    - HAL_DMAEx_List_UnLinkQ()
    - HAL_DMAEx_List_DeInit()

Note: *When a peripheral is used in DMA mode, the DMA initialization must be done in the HAL_PPP_MspInit() callback. In addition, the user application must associate the DMA handle to the PPP handle (refer to section “HAL IO operation functions”).*

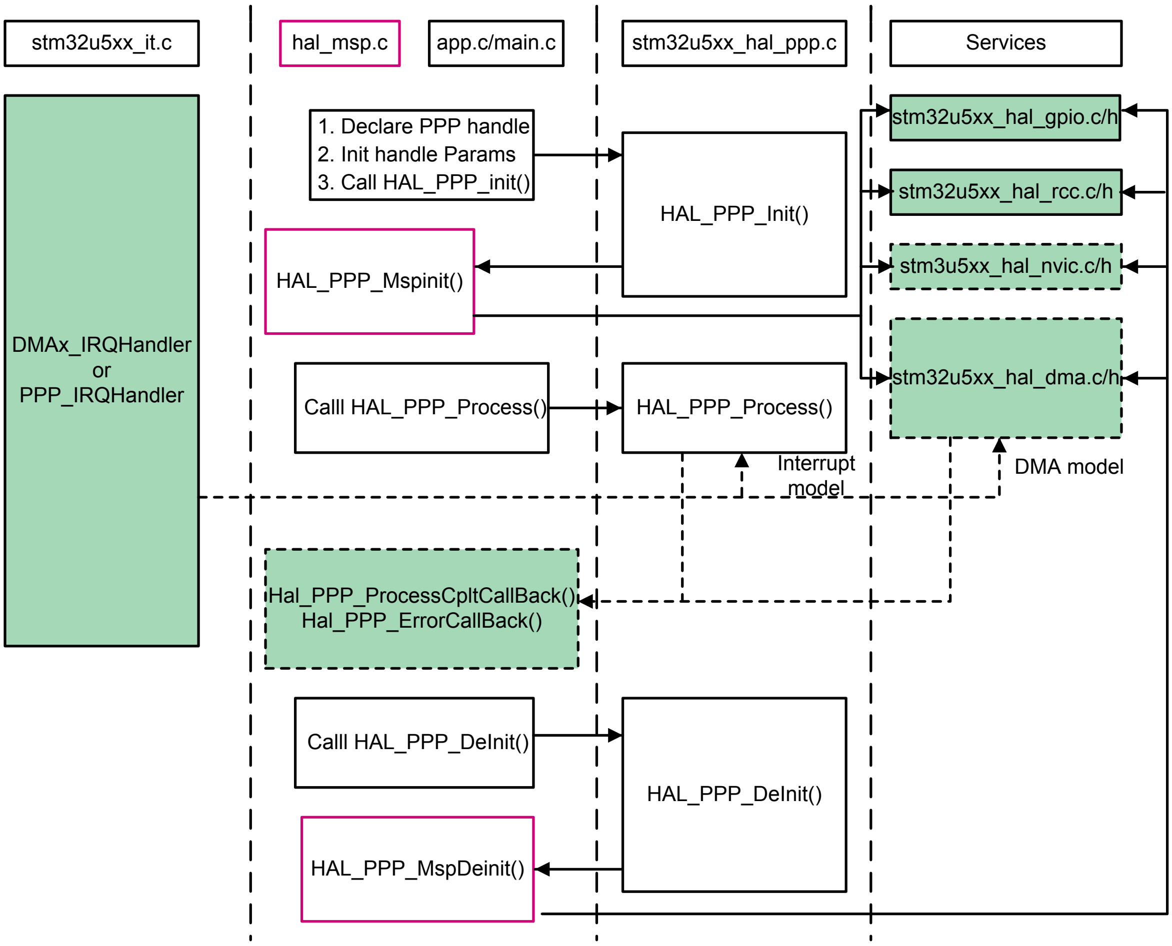
Note: *DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.*

3.12 How to use HAL drivers

3.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 6. HAL driver model



Note: The functions implemented in the HAL driver are shown in green, the functions called from interrupt handlers in dotted lines, and the msp functions implemented in the user application in red. Non-dotted lines represent the interactions between the user application functions.

Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

3.12.2 HAL initialization

3.12.2.1 *HAL global initialization*

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32u5xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as “weak” empty function in the HAL drivers.
- HAL_DeInit()
  - resets all peripherals
  - calls function HAL_MspDeInit() which is a user callback function to do system level De-Initalizations.

- HAL_GetTick(): this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if HAL_Delay() is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR is blocked.

3.12.2.2 *System clock initialization*

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code.

Below the typical clock configuration sequence to reach the maximum clock frequency of 80 MHz based on the HSE clock.

```
 void SystemClock_Config(void)
{
   RCC_ClkInitTypeDef clkinitstruct = {0};
   RCC_OscInitTypeDef oscinitstruct = {0};
/* Configure PLLs------------------------------------------------------*/
/* PLL configuration: PLLCLK =  (HSE/PLLM * PLLN) / PLLR = (16/1 * 20) / 2 = 80 MHz*/
/* Enable HSE Oscillator and activate PLL with HSE as source */
   oscinitstruct.OscillatorType =  RCC_OSCILLATORTYPE_HSE;
   oscinitstruct.HSEState =   RCC_HSE_ON;
   oscinitstruct.PLL.PLLState = RCC_PLL_ON;
   oscinitstruct.PLL.PLLSource =   RCC_PLLSOURCE_HSE;
   oscinitstruct.PLL.PLLM = 1;
   oscinitstruct.PLL.PLLN = 20;
   oscinitstruct.PLL.PLLR = 2;
   oscinitstruct.PLL.PLLL = 7;
   oscinitstruct.PLL.PLLQ = 4;
   if   (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
   {
    /* Initialization Error */
       while(1);
   }
   /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2   clock divid
ers    */
   clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK I   RCC_CLOCKTYPE_HCLK I RCC_CLOCKTYPE_PCL
K1 I RCC_CLOCKTYPE_PCLK2);
   clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
   clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
   clkinitstruct.APB2CLKDivider =   RCC_HCLK_DIV1;
   clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV1;
   if
      (HAL_RCC_ClockConfig(&clkinitstruct,FLASH_LATENCY_4)!= HAL_OK)
    {
      /*   Initialization Error */
     while(1);
    }
 }
```

3.12.2.3 *HAL MSP initialization process*

The peripheral initialization is done through *HAL_PPP_Init()* while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function *HAL_PPP_MspInit().*

The MspInit callback performs the low level initialization related to the different additional hardware resources: RCC, GPIO, NVIC and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and de-initialization:

```
/**
* @brief Initializes the PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspInit could be implemented in the user file */
}
/**
* @brief DeInitializes PPP MSP.
* @param hppp: PPP handle
* @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
/* NOTE : This function Should not be modified, when the callback is needed,
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low level initialization code or omit them and use his own initialization routine.

The HAL MSP callback is implemented inside the *stm32u5xx_hal_msp.c* file in the user folders. An *stm32u5xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32u5xx_hal_msp.c file contains the following functions:

Table 14. **MSP functions**

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP De-Initialization in the *Hal_MspDeInit().* In this case the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be de-initialized in run time and the low level resources of a given peripheral need to be released and used by another peripheral, HAL_PPP_MspDeInit() and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit().*

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

3.12.3 HAL I/O operation process

The HAL functions with internal data processing like transmit, receive, write and read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

3.12.3.1 Polling mode

In Polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical Polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(…)
return HAL_OK; }
```

3.12.3.2 *Interrupt mode*

In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In Interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launches the process
- *HAL_PPP_IRQHandler()*: global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: callback relative to the process Error.

To use a process in Interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and HAL_PPP_IRQHandler in *stm32u5xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

*stm32u5xx_it.c*file:

3.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32u5xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART1;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
__HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
__HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver that means, the user can declare it again in the application code. The function in the driver should not be modified.

An example of use is illustrated below:

main.c file:

stm32u5xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

3.12.4 Timeout and error management

3.12.4.1 Timeout management

The timeout is often used for the APIs that operate in Polling mode. It defines the delay during which a blocking process should wait till an error is returned. An example is provided below:

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t CompleteLevel, ui
nt32_t Timeout)
```

The timeout possible values are the following:

Table 15. Timeout values

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) [(1)]	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

1. *HAL_MAX_DELAY is defined in the stm32u5xx_hal_def.h as 0xFFFFFFFF*

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef *hppp)
{
  (…)
  tickstart = HAL_GetTick();
  (…)
 while (ProcessOngoing)
  {
    (…)
   if ((HAL_GetTick() – tickstart) > LOCAL_PROCESS_TIMEOUT)
    {
      hppp->ErrorCode |= HAL_PPP_ERROR_TIMEOUT;
      hppp->State= HAL_PPP_STATE_READY;
      return HAL_ERROR;
    }
  }
  (…)
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_StatusTypeDef HAL_PPP_Poll(PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
  (…)
  tickstart = HAL_GetTick();
  (…)
 while (ProcessOngoing)
  {
    (…)
   if (((HAL_GetTick() – tickstart) > Timeout) || (Timeout == 0))
    {
      hppp->ErrorCode |= HAL_PPP_ERROR_TIMEOUT;
```

```
      hppp->State= HAL_PPP_STATE_READY;
      return HAL_ERROR;
    }
  }
  (…)
}
```

3.12.4.2 Error management

The HAL drivers implement a check on the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system may crash or go into an undefined state. These critical parameters are checked before being used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32 Size
)
{
if ((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL_StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{
if (hppp == NULL) //the handle should be already allocated
{
return HAL_ERROR;
}
}
```

- Timeout error: the following statement is used when a timeout error occurs:

```
while (Process ongoing)
{
timeout = HAL_GetTick() + Timeout; while (data processing is running)
{
if(timeout) { return HAL_TIMEOUT;
}
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process ()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError ()* to allow retrieving the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
PPP_TypeDef * Instance; /* PPP registers base address */
PPP_InitTypeDef Init; /* PPP initialization parameters */
HAL_LockTypeDef Lock; /* PPP locking object */
__IO HAL_PPP_StateTypeDef State; /* PPP state */
__IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
(…)
/* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY;  /* Set the peripheral ready */
PP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
_HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR;  /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
ErrorCode = HAL_PPP_GetError (hppp); /* retreive error code */
}
```

3.12.4.3 *Run-time checking*

The HAL implements run-time failure detection by checking the input values of all HAL driver functions. The run-time checking is achieved by using an *assert_param* macro. This macro is used in all the HAL driver functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

To enable the run-time checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in *stm32u5xx_hal_conf.h* file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
(..) /* Check the parameters */
assert_param(IS_UART_INSTANCE(huart->Instance));
assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
assert_param(IS_UART_PARITY(huart->Init.Parity));
assert_param(IS_UART_MODE(huart->Init.Mode));
assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
(..)
```

```
/** @defgroup UART_Word_Length *
@{
*/
#define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
#define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
#define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
\ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the assert_param macro is false, the*assert_failed* function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The *assert_param* macro is implemented in stm32u5xx_hal_conf.h:

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr: If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions --------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

The *assert_failed* function is implemented in the main.c file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* Infinite loop */
while (1)
{
```

```
}
}
```

Attention: ***Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.***

4 Overview of low-layer drivers

The low-layer (LL) drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as USB).

The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features

The low-layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content.

4.1 Low-layer files

The low-layer drivers are built around header/C files (one per each supported peripheral) plus five header files for some System and Cortex related features.

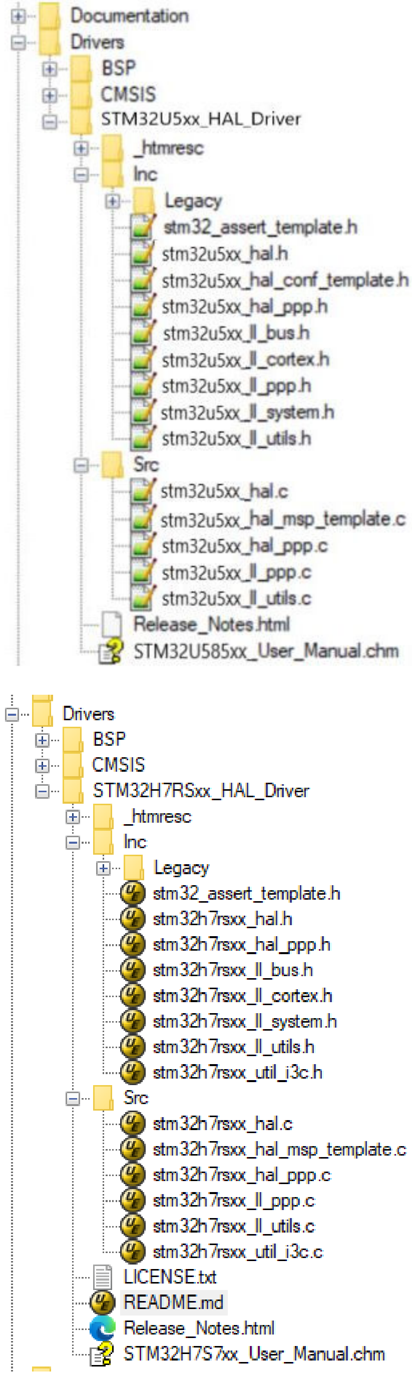
Table 16. **LL driver files**

File	Description
stm32u5xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32u5xx_ll_ppp.h/.c	stm32u5xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are defined within stm32u5xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32u5xx_ll_ppp.h file.
stm32u5xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (such as LL_SYSTICK_xxxxx and LL_LPM_xxxxx "Low Power Mode")
stm32u5xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
stm32u5xx_ll_system.h	System related operations. Example: LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

Note: There is no configuration file for the LL drivers.

The low-layer files are located in the same HAL driver folder.

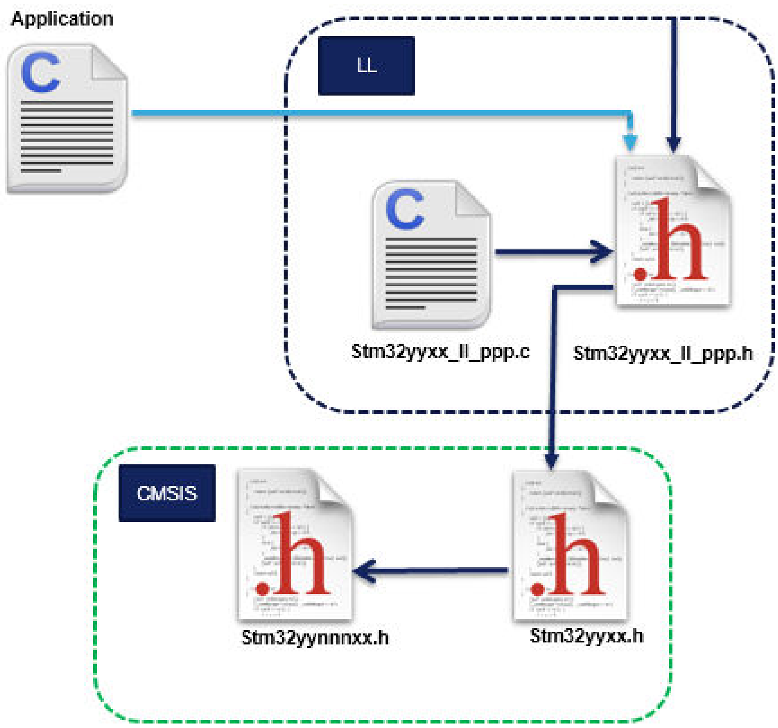
Figure 7. Low-layer driver folders



In general, low-layer drivers include only the STM32 CMSIS device file.

```
#include "stm32yyxx.h"
```

Figure 8. **Low-layer driver CMSIS files**



Application files have to include only the used low-layer driver header files.

4.2 Overview of low-layer APIs and naming rules

4.2.1 Peripheral initialization functions

The LL drivers offer three sets of initialization functions. They are defined in stm32u5xx_ll_ppp.c file:

- Functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Function for peripheral de-initialization (peripheral registers restored to their default values)

The definition of these LL initialization functions and associated resources (structure, literals and prototypes) is conditioned by a compilation switch: *USE_FULL_LL_DRIVER*. To use these functions, this switch must be added in the toolchain compiler preprocessor or to any generic header file which is processed before the LL drivers.

The below table shows the list of the common functions provided for all the supported peripherals:

Table 17. Common peripheral initialization functions

Functions	Return Type	Parameters	Description
LL_PPP_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_InitTypeDef* PPP_InitStruct*	Initializes the peripheral main features according to the parameters specified in PPP_InitStruct. Example: LL_USART_Init(USART_TypeDef *USARTx, LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_StructInit	*void*	• *LL_PPP_InitTypeDef* PPP_InitStruct*	Fills each PPP_InitStruct member with its default value. Example. LL_USART_StructInit(LL_USART_InitTypeDef *USART_InitStruct)
LL_PPP_DeInit	*ErrorStatus*	• *PPP_TypeDef* PPPx*	De-initializes the peripheral registers, that is restore them to their default reset values. Example. LL_USART_DeInit(USART_TypeDef *USARTx)

Additional functions are available for some peripherals (refer to Table 18. Optional peripheral initialization functions).

Table 18. Optional peripheral initialization functions

Functions	Return Type	Parameters	Examples
LL_PPP{_CATEGORY}_Init	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Initializes peripheral features according to the parameters specified in PPP_InitStruct. Example: LL_ADC_INJ_Init(ADC_TypeDef *ADCx, LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct) LL_RTC_TIME_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef *RTC_TimeStruct) LL_RTC_DATE_Init(RTC_TypeDef *RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef *RTC_DateStruct) LL_TIM_IC_Init(TIM_TypeDef* TIMx, uint32_t Channel, LL_TIM_IC_InitTypeDef* TIM_IC_InitStruct) LL_TIM_ENCODER_Init(TIM_TypeDef* TIMx, LL_TIM_ENCODER_InitTypeDef* TIM_EncoderInitStruct)
LL_PPP{_CATEGORY}_StructInit	*void*	*LL_PPP{_CATEGORY}_InitTypeDef* PPP{_CATEGORY}_InitStruct*	Fills each *PPP{_CATEGORY}_InitStruct* member with its default value. Example: LL_ADC_INJ_StructInit(LL_ADC_INJ_InitTypeDef *ADC_INJ_InitStruct)
LL_PPP_CommonInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Initializes the common features shared between different instances of the same peripheral. Example: LL_ADC_CommonInit(ADC_Common_TypeDef *ADCxy_COMMON, LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_CommonStructInit	*void*	*LL_PPP_CommonInitTypeDef* PPP_CommonInitStruct*	Fills each *PPP_CommonInitStruct* member with its default value Example: LL_ADC_CommonStructInit(LL_ADC_CommonInitTypeDef *ADC_CommonInitStruct)
LL_PPP_ClockInit	*ErrorStatus*	• *PPP_TypeDef* PPPx* • *LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Initializes the peripheral clock configuration in synchronous mode. Example: LL_USART_ClockInit(USART_TypeDef *USARTx, LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

Functions	Return Type	Parameters	Examples
LL_PPP_ClockStructInit	*void*	*LL_PPP_ClockInitTypeDef* PPP_ClockInitStruct*	Fills each *PPP_ClockInitStruct* member with its default value Example: LL_USART_ClockStructInit(LL_USART_ClockInitTypeDef *USART_ClockInitStruct)

4.2.1.1 *Run-time checking*

Like HAL drivers, LL initialization functions implement run-time failure detection by checking the input values of all LL driver functions. For more details, refer to Section 3.12.4.3: Run-time checking.

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use run-time checking:

1. Copy stm32_assert_template.h to the application folder and rename it to stm32_assert.h. This file defines the assert_param macro which is used when run-time checking is enabled.
2. Include stm32_assert.h file within the application main header file.
3. Add the USE_FULL_ASSERT compilation switch in the toolchain compiler preprocessor or in any generic header file which is processed before the stm32_assert.h driver.

Note: *Run-time checking is not available for LL inline functions.*

4.2.2 Peripheral register-level configuration functions

On top of the peripheral initialization functions, the LL drivers offer a set of inline functions for direct atomic register access. Their format is as follows:

```
__STATIC_INLINE return_type LL_PPP_Function (PPPx_TypeDef *PPPx, args)
```

The “Function” naming is defined depending to the action category:

- **Specific Interrupt, DMA request and status flags management**: Set/Get/Clear/Enable/Disable flags on interrupt and status registers

Table 19. Specific Interrupt, DMA request and status flags management

Name	Examples
LL_PPP_{_CATEGORY}_ActionItem_BITNAME *LL_PPP{_CATEGORY}_IsItem_BITNAME_Action*	• LL_RCC_IsActiveFlag_LSIRDY • LL_RCC_IsActiveFlag_FWRST() • LL_ADC_ClearFlag_EOC(ADC1) • LL_DMA_ClearFlag_TCx(DMA_TypeDef* DMAx)

Table 20. Available function formats

Item	Action	Format
Flag	Get	*LL_PPP_IsActiveFlag_BITNAME*
	Clear	*LL_PPP_ClearFlag_BITNAME*
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

Note: *BITNAME refers to the peripheral register bit name as described in the product line reference manual.*

- **Peripheral clock activation/deactivation management**: Enable/Disable/Reset a peripheral clock

Table 21. **Peripheral clock activation/deactivation management**

Name	Examples
LL_BUS_GRPx_ActionClock{Mode}	• *LL_AHB2_GRP1_EnableClock (LL_AHB2_GRP1_PERIPH_GPIOA| LL_AHB2_GRP1_PERIPH_GPIOB)* • *LL_APB1_GRP1_EnableClockSleep (LL_APB1_GRP1_PERIPH_DAC1)*

Note: 'x' corresponds to the group index and refers to the index of the modified register on a given bus. 'bus' corresponds to the bus name.

- ***Peripheral activation/deactivation management*** : Enable/disable a peripheral or activate/deactivate specific peripheral features

Table 22. **Peripheral activation/deactivation management**

Name	Examples
LL_PPP{_CATEGORY}_Action{Item} *LL_PPP{_CATEGORY}_IsItemAction*	• *LL_ADC_Enable ()* • *LL_ADC_StartCalibration();* • *LL_ADC_IsCalibrationOnGoing;* • *LL_RCC_HSI_Enable ()* • *LL_RCC_HSI_IsReady()*

- ***Peripheral configuration management*** : Set/get a peripheral configuration settings

Table 23. **Peripheral configuration management**

Name	Examples
LL_PPP{_CATEGORY}_Set{ or Get}ConfigItem	*LL_USART_SetBaudRate (USART2, Clock, LL_USART_BAUDRATE_9600)*

- ***Peripheral register management*** : Write/read the content of a register/retrun DMA relative register address

Table 24. **Peripheral register management**

Name
LL_PPP_WriteReg(__INSTANCE__, __REG__, __VALUE__)
LL_PPP_ReadReg(__INSTANCE__, __REG__)
*LL_PPP_DMA_GetRegAddr (PPP_TypeDef *PPPx,{Sub Instance if any ex: Channel} , {uint32_t Propriety})*

Note: The Propriety is a variable used to identify the DMA transfer direction or the data register type.

5 Cohabiting of HAL and LL

The low-ayer APIs are designed to be used in standalone mode or combined with the HAL. They cannot be automatically used with the HAL for the same peripheral instance. If you use the LL APIs for a specific instance, you can still use the HAL APIs for other instances. Be careful that the low-layer APIs might overwrite some registers which content is mirrored in the HAL handles.

5.1 Low-layer driver used in Standalone mode

The low-layer APIs can be used without calling the HAL driver services. This is done by simply including stm32u5xx_ll_ppp.h in the application files. The LL APIs for a given peripheral are called by executing the same sequence as the one recommended by the programming model in the corresponding product line reference manual. In this case the HAL drivers associated to the used peripheral can be removed from the workspace. However the STM32CubeU5 framework should be used in the same way as in the HAL drivers case which means that System file, startup file and CMSIS should always be used.

Note: When the BSP drivers are included, the used HAL drivers associated with the BSP functions drivers should be included in the workspace, even if they are not used by the application layer.

5.2 Mixed use of low-layer APIs and HAL drivers

In this case the low-layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of low-layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the low-layer services can be used together for the same peripheral instance.
- The low-layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, Flash and GPIO).

Several examples showing how to use HAL and LL in the same application are provided within stm32u5 firmware package (refer to Examples_MIX projects).

Note:

1. *When the HAL Init/DeInit APIs are not used and are replaced by the low-layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.*
2. *When process APIs are not used and the corresponding function is performed through the low-layer APIs, the callbacks are not called and post processing or error management should be done by the user application.*
3. *When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.*

6 HAL System Driver

6.1 HAL Firmware driver API description

The following section lists the various functions of the HAL library.

6.1.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

6.1.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initializes the Flash interface the NVIC allocation and initial clock configuration. It initializes the systick also when timeout is needed and the backup domain when enabled.
- De-Initializes common part of the HAL.
- Configure The time base source to have 1ms time base with a dedicated Tick interrupt priority.
  - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
  - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
  - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  - functions affecting time base configurations are declared as __weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- *HAL_Init()*
- *HAL_DeInit()*
- *HAL_MspInit()*
- *HAL_MspDeInit()*
- *HAL_InitTick()*

6.1.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond
- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier
- Enable/Disable Debug module during SLEEP mode
- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:
- *HAL_IncTick()*
- *HAL_GetTick()*
- *HAL_GetTickPrio()*
- *HAL_SetTickFreq()*
- *HAL_GetTickFreq()*
- *HAL_Delay()*
- *HAL_SuspendTick()*
- *HAL_ResumeTick()*
- *HAL_GetHalVersion()*
- *HAL_GetREVID()*
- *HAL_GetDEVID()*
- *HAL_GetUIDw0()*
- *HAL_GetUIDw1()*
- *HAL_GetUIDw2()*

6.1.4 HAL Debug functions

This section provides functions allowing to:
- Enable/Disable Debug module during STOP0/STOP1/STOP2 modes
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:
- *HAL_DBGMCU_EnableDBGStopMode()*
- *HAL_DBGMCU_DisableDBGStopMode()*
- *HAL_DBGMCU_EnableDBGStandbyMode()*
- *HAL_DBGMCU_DisableDBGStandbyMode()*

6.1.5 HAL SYSCFG configuration functions

This section provides functions allowing to:
- Configure the Voltage reference buffer
- Enable/Disable the Voltage reference buffer
- Enable/Disable the I/O analog switch voltage booster

This section contains the following APIs:
- *HAL_SYSCFG_VREFBUF_VoltageScalingConfig()*
- *HAL_SYSCFG_VREFBUF_HighImpedanceConfig()*
- *HAL_SYSCFG_VREFBUF_TrimmingConfig()*
- *HAL_SYSCFG_EnableVREFBUF()*
- *HAL_SYSCFG_DisableVREFBUF()*
- *HAL_SYSCFG_EnableIOAnalogSwitchBooster()*
- *HAL_SYSCFG_DisableIOAnalogSwitchBooster()*
- *HAL_SYSCFG_EnableIOAnalogSwitchVoltageSelection()*
- *HAL_SYSCFG_DisableIOAnalogSwitchVoltageSelection()*
- *HAL_SYSCFG_EnableSRAMCached()*
- *HAL_SYSCFG_DisableSRAMCached()*
- *HAL_SYSCFG_EnableVddCompensationCell()*
- *HAL_SYSCFG_EnableVddIO2CompensationCell()*
- *HAL_SYSCFG_EnableVddHSPICompensationCell()*
- *HAL_SYSCFG_DisableVddCompensationCell()*
- *HAL_SYSCFG_DisableVddIO2CompensationCell()*
- *HAL_SYSCFG_DisableVddHSPICompensationCell()*

- *HAL_SYSCFG_SRAM2Erase()*
- *HAL_SYSCFG_SetOTGPHYReferenceClockSelection()*
- *HAL_SYSCFG_SetOTGPHYPowerDownConfig()*
- *HAL_SYSCFG_EnableOTGPHY()*
- *HAL_SYSCFG_SetOTGPHYDisconnectThreshold()*
- *HAL_SYSCFG_SetOTGPHYSquelchThreshold()*
- *HAL_SYSCFG_SetOTGPHYPreemphasisCurrent()*

6.1.6 SYSCFG lock functions

This section contains the following APIs:

- *HAL_SYSCFG_Lock()*
- *HAL_SYSCFG_GetLock()*

6.1.7 SYSCFG attributes functions

This section contains the following APIs:

- *HAL_SYSCFG_ConfigAttributes()*
- *HAL_SYSCFG_GetConfigAttributes()*

6.1.8 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

Configure the Flash prefetch, the time base source, NVIC and any required global low level hardware by calling the HAL_MspInit() callback function to be optionally defined in user file stm32u5xx_hal_msp.c.

Return values

- **HAL:** status

Notes

- HAL_Init() function is called at the beginning of program after reset and before the clock configuration.
- In the default implementation the System Timer (SysTick) is used as source of time base. The SysTick configuration is based on MSI clock, as MSI is the clock used after a system Reset and the NVIC configuration is set to Priority group 4. Once done, time base tick starts incrementing: the tick variable counter is incremented each 1ms in the SysTick_Handler() interrupt handler.

HAL_DeInit

Function name

HAL_StatusTypeDef HAL_DeInit (void)

Function description

This function de-Initializes common part of the HAL and stops the systick.

Return values

- **HAL:** status

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initializes the MSP.

Return values

- **None:**

HAL_MspDeInit

Function name

void HAL_MspDeInit (void)

Function description

DeInitializes the MSP.

Return values

- **None:**

HAL_InitTick

Function name

HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

Function description

This function configures the source of the time base.

Parameters

- **TickPriority:** Tick interrupt priority.

Return values

- **HAL:** status

Notes

- This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig().
- In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __weak to be overwritten in case of other implementation in user file.

HAL_IncTick

Function name

void HAL_IncTick (void)

Function description

This function is called to increment a global variable "uwTick" used as application time base.

Return values

- **None:**

Notes

- In the default implementation, this variable is incremented each 1ms in SysTick ISR.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_Delay

Function name

void HAL_Delay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on variable incremented.

Parameters

- **Delay:** specifies the delay time length, in milliseconds.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTick

Function name

uint32_t HAL_GetTick (void)

Function description

Provides a tick value in millisecond.

Return values

- **tick:** value

Notes

- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetTickPrio

Function name

uint32_t HAL_GetTickPrio (void)

Function description

This function returns a tick priority.

Return values

- **tick:** priority

HAL_SetTickFreq

Function name

HAL_StatusTypeDef HAL_SetTickFreq (HAL_TickFreqTypeDef Freq)

Function description

Set new tick Freq.

Return values

- **Status:**

HAL_GetTickFreq

Function name

HAL_TickFreqTypeDef HAL_GetTickFreq (void)

Function description

Return tick frequency.

Return values

- **Tick:** frequency. Value of HAL_TickFreqTypeDef.

HAL_SuspendTick

Function name

void HAL_SuspendTick (void)

Function description

Suspend Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the SysTick interrupt will be disabled and so Tick increment is suspended.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_ResumeTick

Function name

void HAL_ResumeTick (void)

Function description

Resume Tick increment.

Return values

- **None:**

Notes

- In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the SysTick interrupt will be enabled and so Tick increment is resumed.
- This function is declared as __weak to be overwritten in case of other implementations in user file.

HAL_GetHalVersion

Function name

uint32_t HAL_GetHalVersion (void)

Function description

Returns the HAL revision.

Return values

- **version:** : 0xXYZR (8bits for each decimal, R for RC)

HAL_GetREVID

Function name

uint32_t HAL_GetREVID (void)

Function description

Returns the device revision identifier.

Return values

- **Device:** revision identifier

HAL_GetDEVID

Function name

uint32_t HAL_GetDEVID (void)

Function description

Returns the device identifier.

Return values

- **Device:** identifier

HAL_GetUIDw0

Function name

uint32_t HAL_GetUIDw0 (void)

Function description

Return the first word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_GetUIDw1

Function name

uint32_t HAL_GetUIDw1 (void)

Function description

Return the second word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_GetUIDw2

Function name

uint32_t HAL_GetUIDw2 (void)

Function description

Return the third word of the unique device identifier (UID based on 96 bits)

Return values

- **Device:** identifier

HAL_DBGMCU_EnableDBGStopMode

Function name

void HAL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during STOP0/STOP1/STOP2 modes.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name

void HAL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP0/STOP1/STOP2 modes.

Return values

- **None:**

HAL_DBGMCU_EnableDBGStandbyMode

Function name

void HAL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStandbyMode

Function name

void HAL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during STANDBY mode.

Return values

- **None:**

HAL_SYSCFG_SRAM2Erase

Function name

void HAL_SYSCFG_SRAM2Erase (void)

Function description

HAL_SYSCFG_VREFBUF_VoltageScalingConfig

Function name

void HAL_SYSCFG_VREFBUF_VoltageScalingConfig (uint32_t VoltageScaling)

Function description

Configure the internal voltage reference buffer voltage scale.

Parameters

- **VoltageScaling:** specifies the output voltage to achieve This parameter can be one of the following values:
  - SYSCFG_VREFBUF_VOLTAGE_SCALE0: VREF_OUT1 around 1.5 V. This requires VDDA equal to or higher than 1.8 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE1: VREF_OUT1 around 1.8 V. This requires VDDA equal to or higher than 2.1 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE2: VREF_OUT1 around 2.048 V. This requires VDDA equal to or higher than 2.4 V.
  - SYSCFG_VREFBUF_VOLTAGE_SCALE3: VREF_OUT1 around 2.5 V. This requires VDDA equal to or higher than 2.8 V.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_HighImpedanceConfig

Function name

void HAL_SYSCFG_VREFBUF_HighImpedanceConfig (uint32_t Mode)

Function description

Configure the internal voltage reference buffer high impedance mode.

Parameters

- **Mode:** specifies the high impedance mode This parameter can be one of the following values:
  - SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE: VREF+ pin is internally connect to VREFINT output.
  - SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE: VREF+ pin is high impedance.

Return values

- **None:**

HAL_SYSCFG_VREFBUF_TrimmingConfig

Function name

void HAL_SYSCFG_VREFBUF_TrimmingConfig (uint32_t TrimmingValue)

Function description

Tune the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_SYSCFG_EnableVREFBUF

Function name

HAL_StatusTypeDef HAL_SYSCFG_EnableVREFBUF (void)

Function description

Enable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **HAL_OK/HAL_TIMEOUT:**

HAL_SYSCFG_DisableVREFBUF

Function name

void HAL_SYSCFG_DisableVREFBUF (void)

Function description

Disable the Internal Voltage Reference buffer (VREFBUF).

Return values

- **None:**

HAL_SYSCFG_SetOTGPHYReferenceClockSelection

Function name

void HAL_SYSCFG_SetOTGPHYReferenceClockSelection (uint32_t RefClkSelection)

Function description

Set the OTG PHY reference clock selection.

Parameters

- **RefClkSelection:** Defines the OTG PHY reference clock selection. This parameter can be one of the OTG PHY Reference Clock Selection

Return values

- **None:**

HAL_SYSCFG_SetOTGPHYPowerDownConfig

Function name

void HAL_SYSCFG_SetOTGPHYPowerDownConfig (uint32_t PowerDownConfig)

Function description

Set the OTG PHY Power Down config.

Parameters

- **PowerDownConfig:** Defines the OTG PHY Power down configuration. This parameter can be one of OTG PHY Power Down

Return values

- **None:**

HAL_SYSCFG_EnableOTGPHY

Function name

void HAL_SYSCFG_EnableOTGPHY (uint32_t OTGPHYConfig)

Function description

Enable the OTG PHY .

Parameters

- **OTGPHYConfig:** Defines the OTG PHY configuration. This parameter can be one of OTG PHY Enable

Return values

- **None:**

HAL_SYSCFG_SetOTGPHYDisconnectThreshold

Function name

void HAL_SYSCFG_SetOTGPHYDisconnectThreshold (uint32_t DisconnectThreshold)

Function description

Set the OTG PHY Disconnect Threshold.

Parameters

- **DisconnectThreshold:** Defines the voltage level for the threshold used to detect a disconnect event. This parameter can be one of the OTG PHYTUNER Disconnect Threshold

Return values

- **None:**

HAL_SYSCFG_SetOTGPHYSquelchThreshold

Function name

void HAL_SYSCFG_SetOTGPHYSquelchThreshold (uint32_t SquelchThreshold)

Function description

Adjust the voltage level for the threshold used to detect valid high speed data.

Parameters

- **SquelchThreshold:** Defines the voltage level. This parameter can be onez of the OTG PHYTUNER Squelch Threshold

Return values

- **None:**

HAL_SYSCFG_SetOTGPHYPreemphasisCurrent

Function name

void HAL_SYSCFG_SetOTGPHYPreemphasisCurrent (uint32_t PreemphasisCurrent)

Function description

Set the OTG PHY Current config.

Parameters

- **PreemphasisCurrent:** Defines the current configuration. This parameter can be one of the OTG PHYTUNER Preemphasis Current

Return values

- **None:**

HAL_SYSCFG_EnableIOAnalogSwitchBooster

Function name

void HAL_SYSCFG_EnableIOAnalogSwitchBooster (void)

Function description

Enable the I/O analog switch voltage booster.

Return values

- **None:**

HAL_SYSCFG_DisableIOAnalogSwitchBooster

Function name

void HAL_SYSCFG_DisableIOAnalogSwitchBooster (void)

Function description

Disable the I/O analog switch voltage booster.

Return values

- **None:**

HAL_SYSCFG_EnableIOAnalogSwitchVoltageSelection

Function name

void HAL_SYSCFG_EnableIOAnalogSwitchVoltageSelection (void)

Function description

Enable the I/O analog switch voltage selection.

Return values

- **None:**

HAL_SYSCFG_DisableIOAnalogSwitchVoltageSelection

Function name

void HAL_SYSCFG_DisableIOAnalogSwitchVoltageSelection (void)

Function description

Disable the I/O analog switch voltage selection.

Return values

- **None:**

HAL_SYSCFG_EnableSRAMCached

Function name

void HAL_SYSCFG_EnableSRAMCached (void)

Function description

Enable the Cacheability of internal SRAMx by DCACHE2.

Return values

- **None:**

HAL_SYSCFG_DisableSRAMCached

Function name

void HAL_SYSCFG_DisableSRAMCached (void)

Function description

Disable the Cacheability of internal SRAMx by DCACHE2.

Return values

- **None:**

HAL_SYSCFG_EnableVddCompensationCell

Function name

void HAL_SYSCFG_EnableVddCompensationCell (void)

Function description

Enable the Compensation Cell of GPIO supplied by VDD.

Return values

- **None:**

Notes

- The vdd compensation cell can be used only when the device supply voltage ranges from 1.71 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN1 HAL_SYSCFG_EnableVddCompensationCell

HAL_SYSCFG_EnableVddIO2CompensationCell

Function name

void HAL_SYSCFG_EnableVddIO2CompensationCell (void)

Function description

Enable the Compensation Cell of GPIO supplied by VDDIO2.

Return values

- **None:**

Notes

- The Vdd I/O compensation cell can be used only when the device supply voltage ranges from 1.08 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN2 HAL_SYSCFG_EnableVddIO2CompensationCell

HAL_SYSCFG_EnableVddHSPICompensationCell

Function name

void HAL_SYSCFG_EnableVddHSPICompensationCell (void)

Function description

Enable the Compensation Cell of HSPI IO supplied by VDD.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR EN3 HAL_SYSCFG_EnableVddHSPICompensationCell

HAL_SYSCFG_DisableVddCompensationCell

Function name

void HAL_SYSCFG_DisableVddCompensationCell (void)

Function description

Disable the Compensation Cell of GPIO supplied by VDD.

Return values

- **None:**

Notes

- The Vdd compensation cell can be used only when the device supply voltage ranges from 1.71 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN1 HAL_SYSCFG_DisableVddCompensationCell

HAL_SYSCFG_DisableVddIO2CompensationCell

Function name

void HAL_SYSCFG_DisableVddIO2CompensationCell (void)

Function description

Disable the Compensation Cell of GPIO supplied by VDDIO2.

Return values

- **None:**

Notes

- The Vdd I/O compensation cell can be used only when the device supply voltage ranges from 1.08 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN2 HAL_SYSCFG_DisableVddIO2CompensationCell

HAL_SYSCFG_DisableVddHSPICompensationCell

Function name

void HAL_SYSCFG_DisableVddHSPICompensationCell (void)

Function description

Disable the Compensation Cell of HSPI IO supplied by VDD.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR EN3 HAL_SYSCFG_DisableVddHSPICompensationCell

HAL_SYSCFG_Lock

Function name

void HAL_SYSCFG_Lock (uint32_t Item)

Function description

Lock the SYSCFG item(s).

Parameters

- **Item:** Item(s) to set lock on. This parameter can be a combination of SYSCFG Lock items

Return values

- **None:**

Notes

- Setting lock(s) depends on privilege mode in secure/non-secure code Lock(s) cleared only at system reset

HAL_SYSCFG_GetLock

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetLock (uint32_t * pItem)

Function description

Get the lock state of SYSCFG item.

Parameters

- **pItem:** pointer to return locked items the return value can be a combination of SYSCFG Lock items

Return values

- **HAL:** status

Notes

- Getting lock(s) depends on privilege mode in secure/non-secure code

HAL_SYSCFG_ConfigAttributes

Function name

void HAL_SYSCFG_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the SYSCFG item attribute(s).

Parameters

- **Item:** Item(s) to set attributes on. This parameter can be a one or a combination of SYSCFG Attributes items
- **Attributes:** specifies the secure/non-secure attributes.

Return values

- **None:**

Notes

- Available attributes are to secure SYSCFG items, so this function is only available in secure

HAL_SYSCFG_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_SYSCFG_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get the attribute of a SYSCFG item.

Parameters

- **Item:** Single item to get secure/non-secure attribute from.
- **pAttributes:** pointer to return the attribute.

Return values

- **HAL:** status

Notes

- Available attributes are to secure SYSCFG items, so this function is only available in secure

6.2 HAL Firmware driver defines

The following section lists the various define and macros of the module.

6.2.1 HAL

HAL

DBGMCU Exported Macros

__HAL_DBGMCU_FREEZE_TIM2

__HAL_DBGMCU_UNFREEZE_TIM2

__HAL_DBGMCU_FREEZE_TIM3

__HAL_DBGMCU_UNFREEZE_TIM3

__HAL_DBGMCU_FREEZE_TIM4

__HAL_DBGMCU_UNFREEZE_TIM4

__HAL_DBGMCU_FREEZE_TIM5

__HAL_DBGMCU_UNFREEZE_TIM5

__HAL_DBGMCU_FREEZE_TIM6

__HAL_DBGMCU_UNFREEZE_TIM6

__HAL_DBGMCU_FREEZE_TIM7

__HAL_DBGMCU_UNFREEZE_TIM7

__HAL_DBGMCU_FREEZE_WWDG

__HAL_DBGMCU_UNFREEZE_WWDG

__HAL_DBGMCU_FREEZE_IWDG

__HAL_DBGMCU_UNFREEZE_IWDG

__HAL_DBGMCU_FREEZE_I2C1

__HAL_DBGMCU_UNFREEZE_I2C1

__HAL_DBGMCU_FREEZE_I2C2

__HAL_DBGMCU_UNFREEZE_I2C2

__HAL_DBGMCU_FREEZE_I2C4

__HAL_DBGMCU_UNFREEZE_I2C4

__HAL_DBGMCU_FREEZE_I2C5

__HAL_DBGMCU_UNFREEZE_I2C5

__HAL_DBGMCU_FREEZE_I2C6

__HAL_DBGMCU_UNFREEZE_I2C6

__HAL_DBGMCU_FREEZE_LPTIM2

__HAL_DBGMCU_UNFREEZE_LPTIM2

__HAL_DBGMCU_FREEZE_TIM1

__HAL_DBGMCU_UNFREEZE_TIM1

__HAL_DBGMCU_FREEZE_TIM8

__HAL_DBGMCU_UNFREEZE_TIM8

__HAL_DBGMCU_FREEZE_TIM15

__HAL_DBGMCU_UNFREEZE_TIM15

__HAL_DBGMCU_FREEZE_TIM16

__HAL_DBGMCU_UNFREEZE_TIM16

__HAL_DBGMCU_FREEZE_TIM17

__HAL_DBGMCU_UNFREEZE_TIM17

__HAL_DBGMCU_FREEZE_I2C3

__HAL_DBGMCU_UNFREEZE_I2C3

__HAL_DBGMCU_FREEZE_LPTIM1

__HAL_DBGMCU_UNFREEZE_LPTIM1

__HAL_DBGMCU_FREEZE_LPTIM3

__HAL_DBGMCU_UNFREEZE_LPTIM3

__HAL_DBGMCU_FREEZE_LPTIM4

__HAL_DBGMCU_UNFREEZE_LPTIM4

__HAL_DBGMCU_FREEZE_RTC

__HAL_DBGMCU_UNFREEZE_RTC

__HAL_DBGMCU_FREEZE_GPDMA0

__HAL_DBGMCU_UNFREEZE_GPDMA0

__HAL_DBGMCU_FREEZE_GPDMA1

__HAL_DBGMCU_UNFREEZE_GPDMA1

__HAL_DBGMCU_FREEZE_GPDMA2

__HAL_DBGMCU_UNFREEZE_GPDMA2

__HAL_DBGMCU_FREEZE_GPDMA3

__HAL_DBGMCU_UNFREEZE_GPDMA3

__HAL_DBGMCU_FREEZE_GPDMA4

__HAL_DBGMCU_UNFREEZE_GPDMA4

__HAL_DBGMCU_FREEZE_GPDMA5

__HAL_DBGMCU_UNFREEZE_GPDMA5

__HAL_DBGMCU_FREEZE_GPDMA6

__HAL_DBGMCU_UNFREEZE_GPDMA6

__HAL_DBGMCU_FREEZE_GPDMA7

__HAL_DBGMCU_UNFREEZE_GPDMA7

__HAL_DBGMCU_FREEZE_GPDMA8

__HAL_DBGMCU_UNFREEZE_GPDMA8

__HAL_DBGMCU_FREEZE_GPDMA9

__HAL_DBGMCU_UNFREEZE_GPDMA9

__HAL_DBGMCU_FREEZE_GPDMA10

__HAL_DBGMCU_UNFREEZE_GPDMA10

__HAL_DBGMCU_FREEZE_GPDMA11

__HAL_DBGMCU_UNFREEZE_GPDMA11

__HAL_DBGMCU_FREEZE_GPDMA12

__HAL_DBGMCU_UNFREEZE_GPDMA12

__HAL_DBGMCU_FREEZE_GPDMA13

__HAL_DBGMCU_UNFREEZE_GPDMA13

__HAL_DBGMCU_FREEZE_GPDMA14

__HAL_DBGMCU_UNFREEZE_GPDMA14

__HAL_DBGMCU_FREEZE_GPDMA15

__HAL_DBGMCU_UNFREEZE_GPDMA15

__HAL_DBGMCU_FREEZE_LPDMA0

__HAL_DBGMCU_UNFREEZE_LPDMA0

__HAL_DBGMCU_FREEZE_LPDMA1

__HAL_DBGMCU_UNFREEZE_LPDMA1

__HAL_DBGMCU_FREEZE_LPDMA2

__HAL_DBGMCU_UNFREEZE_LPDMA2

__HAL_DBGMCU_FREEZE_LPDMA3

__HAL_DBGMCU_UNFREEZE_LPDMA3

HAL FDCAN Error Code

HAL_FDCAN_ERROR_NONE

No error

HAL_FDCAN_ERROR_TIMEOUT

Timeout error

HAL_FDCAN_ERROR_NOT_INITIALIZED

Peripheral not initialized

HAL_FDCAN_ERROR_NOT_READY

Peripheral not ready

HAL_FDCAN_ERROR_NOT_STARTED

Peripheral not started

HAL_FDCAN_ERROR_NOT_SUPPORTED

Mode not supported

HAL_FDCAN_ERROR_PARAM

Parameter error

HAL_FDCAN_ERROR_PENDING

Pending operation

HAL_FDCAN_ERROR_RAM_ACCESS

Message RAM Access Failure

HAL_FDCAN_ERROR_FIFO_EMPTY

Put element in full FIFO

HAL_FDCAN_ERROR_FIFO_FULL

Get element from empty FIFO

HAL_FDCAN_ERROR_LOG_OVERFLOW

Overflow of CAN Error Logging Counter

HAL_FDCAN_ERROR_RAM_WDG

Message RAM Watchdog event occurred

HAL_FDCAN_ERROR_PROTOCOL_ARBT

Protocol Error in Arbitration Phase (Nominal Bit Time is used)

HAL_FDCAN_ERROR_PROTOCOL_DATA

Protocol Error in Data Phase (Data Bit Time is used)

HAL_FDCAN_ERROR_RESERVED_AREA

Access to Reserved Address

HAL_FDCAN_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL state definition

HAL_SMBUS_STATE_RESET

SMBUS not yet initialized or disabled

HAL_SMBUS_STATE_READY

SMBUS initialized and ready for use

HAL_SMBUS_STATE_BUSY

SMBUS internal process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_TX

Master Data Transmission process is ongoing

HAL_SMBUS_STATE_MASTER_BUSY_RX

Master Data Reception process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_TX

Slave Data Transmission process is ongoing

HAL_SMBUS_STATE_SLAVE_BUSY_RX

Slave Data Reception process is ongoing

HAL_SMBUS_STATE_LISTEN

Address Listen Mode is ongoing

SYSCFG attributes

SYSCFG_SEC

Secure attribute

SYSCFG_NSEC

Non-secure attribute

SYSCFG Attributes items

SYSCFG_CLK

SYSCFG clock control

SYSCFG_CLASSB

Class B

SYSCFG_FPU

FPU

SYSCFG_ALL

All

SYSCFG Exported Macros

__HAL_SYSCFG_FPU_INTERRUPT_ENABLE

Description:

- Floating Point Unit interrupt enable/disable macros.

Parameters:

- __INTERRUPT__: This parameter can be a value of

__HAL_SYSCFG_FPU_INTERRUPT_DISABLE

__HAL_SYSCFG_BREAK_ECC_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset.

Enable and lock the connection of Flash ECC error connection to TIM1/8/15/16/17 Break input.

__HAL_SYSCFG_BREAK_LOCKUP_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset.

Enable and lock the connection of Cortex-M33 LOCKUP (Hardfault) output to TIM1/8/15/16/17 Break input.

__HAL_SYSCFG_BREAK_PVD_LOCK

Notes:

- The selected configuration is locked and can be unlocked only by system reset.

Enable and lock the PVD connection to Timer1/8/15/16/17 Break input, as well as the PVDE and PLS[2:0] in the PWR_CR2 register.

__HAL_SYSCFG_BREAK_SRAM2PARITY_LOCK

Notes:

- The selected configuration is locked and can be unlocked by system reset.

Enable and lock the SRAM2 parity error signal connection to TIM1/8/15/16/17 Break input.

__HAL_SYSCFG_GET_FLAG

Description:

- Check SYSCFG flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SYSCFG_FLAG_SRAM2_PE SRAM2 Parity Error Flag
  - SYSCFG_FLAG_SRAM2_BUSY SRAM2 Erase Ongoing

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SYSCFG_CLEAR_FLAG

__HAL_SYSCFG_FASTMODEPLUS_ENABLE

Description:

- Fast-mode Plus driving capability enable/disable macros.

Parameters:

- __FASTMODEPLUS__: This parameter can be a value of :
  - SYSCFG_FASTMODEPLUS_PB6 Fast-mode Plus driving capability activation on PB6
  - SYSCFG_FASTMODEPLUS_PB7 Fast-mode Plus driving capability activation on PB7
  - SYSCFG_FASTMODEPLUS_PB8 Fast-mode Plus driving capability activation on PB8
  - SYSCFG_FASTMODEPLUS_PB9 Fast-mode Plus driving capability activation on PB9

__HAL_SYSCFG_FASTMODEPLUS_DISABLE

Fast-mode Plus on GPIO

SYSCFG_FASTMODEPLUS_PB6

Enable Fast-mode Plus on PB6

SYSCFG_FASTMODEPLUS_PB7

Enable Fast-mode Plus on PB7

SYSCFG_FASTMODEPLUS_PB8

Enable Fast-mode Plus on PB8

SYSCFG_FASTMODEPLUS_PB9

Enable Fast-mode Plus on PB9

Flags

SYSCFG_FLAG_SRAM2_PE

SRAM2 parity error

SYSCFG_FLAG_SRAM2_BUSY

SRAM2 busy by erase operation

FPU Interrupts

SYSCFG_IT_FPU_IOC

Floating Point Unit Invalid operation Interrupt

SYSCFG_IT_FPU_DZC

Floating Point Unit Divide-by-zero Interrupt

SYSCFG_IT_FPU_UFC

Floating Point Unit Underflow Interrupt

SYSCFG_IT_FPU_OFC

Floating Point Unit Overflow Interrupt

SYSCFG_IT_FPU_IDC

Floating Point Unit Input denormal Interrupt

SYSCFG_IT_FPU_IXC

Floating Point Unit Inexact Interrupt

SYSCFG Lock items

SYSCFG_MPU_NSEC

Non-secure MPU lock (privileged secure or non-secure only)

SYSCFG_VTOR_NSEC

Non-secure VTOR lock (privileged secure or non-secure only)

SYSCFG_SAU

SAU lock (privileged secure code only)

SYSCFG_MPU_SEC

Secure MPU lock (privileged secure code only)

SYSCFG_VTOR_AIRCR_SEC

VTOR_S and AIRCR lock (privileged secure code only)

SYSCFG_LOCK_ALL

All

OTG PHYTUNER Disconnect Threshold

SYSCFG_OTG_HS_PHY_DISCONNECT_5_9PERCENT

+5.9% (recommended value)

SYSCFG_OTG_HS_PHY_DISCONNECT_0PERCENT

0% (default value)

OTG PHYTUNER Preemphasis Current

SYSCFG_OTG_HS_PHY_PREEMP_DISABLED

HS transmitter preemphasis circuit disabled

SYSCFG_OTG_HS_PHY_PREEMP_1X

HS transmitter preemphasis circuit sources 1x preemphasis current

SYSCFG_OTG_HS_PHY_PREEMP_2X

HS transmitter preemphasis circuit sources 2x preemphasis current

SYSCFG_OTG_HS_PHY_PREEMP_3X

HS transmitter preemphasis circuit sources 3x preemphasis current

OTG PHYTUNER Squelch Threshold

SYSCFG_OTG_HS_PHY_SQUELCH_15PERCENT

+15% (recommended value)

SYSCFG_OTG_HS_PHY_SQUELCH_0PERCENT

0% (default value)

OTG PHY Enable

SYSCFG_OTG_HS_PHY_UNDERRESET

PHY under reset

SYSCFG_OTG_HS_PHY_ENABLE

PHY enabled

OTG PHY Power Down

SYSCFG_OTG_HS_PHY_POWER_ON

PHY state machine, bias and OTG PHY PLL are powered down

SYSCFG_OTG_HS_PHY_POWER_DOWN

PHY state machine, bias and OTG PHY PLL remain powered

OTG PHY Reference Clock Selection

SYSCFG_OTG_HS_PHY_CLK_SELECT_1

16Mhz

SYSCFG_OTG_HS_PHY_CLK_SELECT_2

19.2Mhz

SYSCFG_OTG_HS_PHY_CLK_SELECT_3

20Mhz

SYSCFG_OTG_HS_PHY_CLK_SELECT_4

24Mhz

SYSCFG_OTG_HS_PHY_CLK_SELECT_5

26Mhz

SYSCFG_OTG_HS_PHY_CLK_SELECT_6

32Mhz

VREFBUF High Impedance

SYSCFG_VREFBUF_HIGH_IMPEDANCE_DISABLE

VREF_plus pin is internally connected to Voltage reference buffer output

SYSCFG_VREFBUF_HIGH_IMPEDANCE_ENABLE

VREF_plus pin is high impedance

VREFBUF Voltage Scale

SYSCFG_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREF_OUT1)

SYSCFG_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREF_OUT2)

SYSCFG_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 2 (VREF_OUT3)

SYSCFG_VREFBUF_VOLTAGE_SCALE3

Voltage reference scale 3 (VREF_OUT4)

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_OversamplingTypeDef

ADC_OversamplingTypeDef is defined in the stm32u5xx_hal_adc.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***
- ***uint32_t TriggeredMode***
- ***uint32_t OversamplingStopReset***

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. In case of ADC1 or ADC2 (if available), this parameter can be in the range from 0 to 1023 In case of ADC4, this parameter can be a value of ***ADC_HAL_EC_OVS_RATIO***
- ***uint32_t ADC_OversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***
- ***uint32_t ADC_OversamplingTypeDef::TriggeredMode***
  Selects the regular triggered oversampling mode. This parameter can be a value of ***ADC_HAL_EC_OVS_DISCONT_MODE***
- ***uint32_t ADC_OversamplingTypeDef::OversamplingStopReset***
  Selects the regular oversampling mode. The oversampling is either temporary stopped or reset upon an injected sequence interruption. If oversampling is enabled on both regular and injected groups, this parameter is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE" (the oversampling buffer is zeroed during injection sequence). This parameter can be a value of ***ADC_HAL_EC_OVS_SCOPE_REG*** Note: This parameter is not applicable for ADC4

7.1.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32u5xx_hal_adc.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t Resolution***
- ***uint32_t GainCompensation***
- ***uint32_t ScanConvMode***
- ***uint32_t DataAlign***
- ***uint32_t EOCSelection***
- ***FunctionalState LowPowerAutoWait***
- ***uint32_t LowPowerAutoPowerOff***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***FunctionalState DiscontinuousConvMode***
- ***uint32_t NbrOfDiscConversion***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***uint32_t ConversionDataManagement***
- ***FunctionalState DMAContinuousRequests***
- ***uint32_t Overrun***
- ***uint32_t SamplingTimeCommon1***
- ***uint32_t SamplingTimeCommon2***

- ***uint32_t LeftBitShift***
- ***FunctionalState OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***
- ***uint32_t TriggerFrequencyMode***
- ***uint32_t VrefProtection***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ClockPrescaler***
  Select ADC clock source (synchronous clock derived from APB clock or asynchronous clock derived from system clock or PLL (Refer to reference manual for list of clocks available)) and clock prescaler. This parameter can be a value of ***ADC_HAL_EC_COMMON_CLOCK_SOURCE***. Note: The ADC clock configuration is common to all ADC instances. Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: In case of synchronous clock mode based on HCLK/1, the configuration must be enabled only if the system clock has a 50% duty clock cycle (APB prescaler configured inside RCC must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference manual for details. Note: In case of usage of asynchronous clock, the selected clock must be preliminarily enabled at RCC top level. Note: This parameter can be modified only if all ADC instances are disabled.
- ***uint32_t ADC_InitTypeDef::Resolution***
  Configure the ADC resolution. This parameter can be a value of ***ADC_HAL_EC_RESOLUTION***
- ***uint32_t ADC_InitTypeDef::GainCompensation***
  Specify the ADC gain compensation coefficient to be applied to ADC raw conversion data, based on following formula: DATA = DATA(raw) * (gain compensation coef) / 4095 14 bit format, unsigned: 2 bits exponents / 14 bits mantissa Gain step is 1/4095 = 0.000244 Gain range is 0.0000 to 3.999756 This parameter value can be 0 Gain compensation will be disabled and coefficient set to 0 1 -> 0x3FFF Gain compensation will be enabled and coefficient set to specified value
- ***uint32_t ADC_InitTypeDef::ScanConvMode***
  Configure the sequencer of ADC groups regular and injected. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. For ADC1 or ADC2 instances, this parameter could be enabled or disabled. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by 'NbrOfConversion' or 'InjectedNbrOfConversion' and rank of each channel in sequencer). Scan direction is upward: from rank 1 to rank 'n'. For ADC4 instance, this parameter could be "fully configurable" or "not fully configurable":
  - sequencer configured to fully configurable: sequencer length and each rank affectation to a channel are configurable.
    - Sequence length: Set number of ranks in the scan sequence.
    - Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
  - sequencer configured to not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number.
    - Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).
    - Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). This parameter can be a value of ***ADC_Scan_mode***
- ***uint32_t ADC_InitTypeDef::DataAlign***
  Specify ADC data alignment in conversion data register (right or left). Refer to reference manual for alignments formats versus resolutions. This parameter can be a value of ***ADC_HAL_EC_DATA_ALIGN***
- ***uint32_t ADC_InitTypeDef::EOCSelection***
  Specify which EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of unitary conversion or end of sequence conversions. This parameter can be a value of ***ADC_EOCSelection***.

- ***FunctionalState ADC_InitTypeDef::LowPowerAutoWait***
  Select the dynamic low power Auto Delay: new conversion start only when the previous conversion (for ADC group regular) or previous sequence (for ADC group injected) has been retrieved by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEx_InjectedGetValue()**. This feature automatically adapts the frequency of ADC conversions triggers to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be set to ENABLE or DISABLE. Note: It is not recommended to use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait). Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when ADC conversion data is needed: use **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start. (in case of usage of ADC group injected, use the equivalent functions HAL_ADCExInjected_Start(), **HAL_ADCEx_InjectedGetValue()**, ...).
- ***uint32_t ADC_InitTypeDef::LowPowerAutoPowerOff***
  Select the auto-off low power mode: the ADC automatically powers-off after a conversion and automatically wakes-up when a new conversion is triggered. This feature can be combined with automatic wait mode ('LowPowerAutoWait'). This parameter can be value of ***ADC_HAL_LowPower_DPD***. Note: If activated, this feature also turns off the ADC dedicated 16 MHz RC oscillator (HSI16). Note: This parameter can be used for ADC4 only.
- ***FunctionalState ADC_InitTypeDef::ContinuousConvMode***
  Specify whether the conversion is performed in single mode (one conversion) or continuous mode for ADC group regular, after the first ADC conversion start trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
  Specify the number of ranks that will be converted within the regular group sequencer. This parameter is dependent on ADC instance. If ADC1 or ADC2 instances: To use the regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 16. If ADC4 instance: This parameter is dependent on ScanConvMode:
  – sequencer configured to fully configurable: Number of ranks in the scan sequence is configurable using this parameter. Note: After the first call of '**HAL_ADC_Init()**', each rank corresponding to parameter "NbrOfConversion" must be set using '**HAL_ADC_ConfigChannel()**'. Afterwards, when all needed sequencer ranks are set, parameter 'NbrOfConversion' can be updated without modifying configuration of sequencer ranks (sequencer ranks above 'NbrOfConversion' are discarded).
  – sequencer configured to not fully configurable: Number of ranks in the scan sequence is defined by number of channels set in the sequence. This parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8. Note: This parameter must be modified when no conversion is on going on regular group (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_InitTypeDef::DiscontinuousConvMode***
  Specify whether the conversions sequence of ADC group regular is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
  Specifies the number of discontinuous conversions in which the main sequence of ADC group regular (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
  Select the external event source used to trigger ADC group regular conversion start. If set to ADC_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_regular_external_trigger_source***. Caution: external trigger source is common to all ADC instances.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
  Select the external event edge used to trigger ADC group regular conversion start. If trigger source is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of ***ADC_regular_external_trigger_edge***

- ***uint32_t ADC_InitTypeDef::ConversionDataManagement***
  Specifies whether the Data conversion data is managed: using the DMA (oneshot or circular), or stored in the DR register or transferred to MDF register. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be a value of ***ADC_ConversionDataManagement***. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_InitTypeDef::DMAContinuousRequests***
  Specify whether the DMA requests are performed in one shot mode (DMA transfer stops when number of conversions is reached) or in continuous mode (DMA transfer unlimited, whatever number of conversions). This parameter can be set to ENABLE or DISABLE. Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached.
- ***uint32_t ADC_InitTypeDef::Overrun***
  Select the behavior in case of overrun: data overwritten or preserved (default). This parameter applies to ADC group regular only. This parameter can be a value of ***ADC_HAL_EC_REG_OVR_DATA_BEHAVIOR***. Note: In case of overrun set to data preserved and usage with programming model with interruption (HAL_Start_IT()): ADC IRQ handler has to clear end of conversion flags, this induces the release of the preserved data. If needed, this data can be saved in function **HAL_ADC_ConvCpltCallback()**, placed in user program code (called before end of conversion flags clear). Note: Error reporting with respect to the conversion mode:
  – Usage with ADC conversion by polling for event or interruption: Error is reported only if overrun is set to data preserved. If overrun is set to data overwritten, user can willingly not read all the converted data, this is not considered as an erroneous case.
  – Usage with ADC conversion by DMA: Error is reported whatever overrun setting (DMA is expected to process all data from data register).
- ***uint32_t ADC_InitTypeDef::SamplingTimeCommon1***
  Set sampling time common to a group of channels. Unit: ADC clock cycles. This parameter is applied for ADC4. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits). Note: On this STM32 family, two different sampling time settings are available, each channel can use one of these two settings. On some other STM32 devices, this parameter in channel wise and is located into ADC channel initialization structure. This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: few tens of microseconds).
- ***uint32_t ADC_InitTypeDef::SamplingTimeCommon2***
  Set sampling time common to a group of channels, second common setting possible. Unit: ADC clock cycles This parameter is applied for ADC4. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits). Note: On this STM32 family, two different sampling time settings are available, each channel can use one of these two settings. On some other STM32 devices, this parameter in channel wise and is located into ADC channel initialization structure. This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME*** Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_vbat, TS_temp (values rough order: few tens of microseconds).
- ***uint32_t ADC_InitTypeDef::LeftBitShift***
  Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADCEx_Left_Bit_Shift***
- ***FunctionalState ADC_InitTypeDef::OversamplingMode***
  Specify whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing on ADC groups regular and injected
- ***ADC_OversamplingTypeDef ADC_InitTypeDef::Oversampling***
  Specify the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling is already enabled.

- ***uint32_t ADC_InitTypeDef::TriggerFrequencyMode***
  Set ADC trigger frequency mode. This parameter can be a value of ***ADC_HAL_EC_REG_TRIGGER_FREQ***. Note: ADC trigger frequency mode must be set to low frequency when a duration is exceeded before ADC conversion start trigger event (between ADC enable and ADC conversion start trigger event or between two ADC conversion start trigger event). Duration value: Refer to device datasheet, parameter "tIdle". Note: When ADC trigger frequency mode is set to low frequency, some rearm cycles are inserted before performing ADC conversion start, inducing a delay of 2 ADC clock cycles.
- ***uint32_t ADC_InitTypeDef::VrefProtection***
  Select the Vref protection mode: specify VREF+ protection mode when multiple ADCs are working simultaneously. This parameter can be value of ***ADC_HAL_VrefProt***. Note: This parameter can be used for ADC4 only.

7.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32u5xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***
- ***uint32_t SingleDiff***
- ***uint32_t OffsetNumber***
- ***uint32_t Offset***
- ***FunctionalState OffsetRightShift***
- ***FunctionalState OffsetSignedSaturation***
- ***FunctionalState OffsetSaturation***
- ***uint32_t OffsetSign***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specify the channel to configure into ADC regular group. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Specify the rank in the regular group sequencer. This parameter is depending on ADC instances. If ADC1 or ADC2 instances, this parameter specify the rank in the sequencer. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted) If ADC4 instance, this parameter allows to add or remove the channel from the ADC regular group sequencer and specify its conversion rank. This parameter is depending on ScanConvMode:
  – sequencer configured to fully configurable: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
  – sequencer configured to not fully configurable: rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Despite the channel rank is fixed, this parameter allow an additional possibility: to remove the selected rank (selected channel) from sequencer. This parameter can be a value of ***ADC_HAL_EC_REG_SEQ_RANKS***
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***
  Sampling time value to be set for the selected channel. Unit: ADC clock cycles Conversion time is the addition of sampling time and processing time (14.5 ADC clock cycles at ADC resolution 14 bits, 12.5 cycles at 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). For ADC1 and 2 (if available): This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used into regular and/or injected group. It overwrites the last setting. For ADC4: This parameter can be a value of ***ADC_HAL_EC_SAMPLINGTIME_COMMON***. Note: On this STM32 family, two different sampling time settings are available (refer to parameters "SamplingTimeCommon1" and "SamplingTimeCommon2"), each channel can use one of these two settings. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values.

- ***uint32_t ADC_ChannelConfTypeDef::SingleDiff***
  Select single-ended or differential input. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING*** Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_ChannelConfTypeDef::OffsetNumber***
  Select the offset number This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB*** Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_ChannelConfTypeDef::Offset***
  Define the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (14, 12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0x3FFF, 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on without both regular and injected groups (ADC disabled, or ADC enabled continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_ChannelConfTypeDef::OffsetRightShift***
  Define the Right-shift data after Offset correction. This parameter is applied only for 14-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE.
- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSignedSaturation***
  Specify whether the Signed saturation feature is used or not. This parameter is only applied when OffsetSaturation is ENABLE. This parameter is applied only for 14-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE. Note:
  – If OffsetSignedSaturation is set to DISABLE the unsigned saturation feature is used
- ***FunctionalState ADC_ChannelConfTypeDef::OffsetSaturation***
  Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note:
  – This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
  – Applicable for ADC1
- ***uint32_t ADC_ChannelConfTypeDef::OffsetSign***
  Define if the offset should be subtracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note:
  – This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
  – Applicable for ADC1

7.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32u5xx_hal_adc.h

Data Fields

- ***uint32_t WatchdogNumber***
- ***uint32_t WatchdogMode***
- ***uint32_t Channel***
- ***FunctionalState ITMode***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***
- ***uint32_t FilteringConfig***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
  Select which ADC analog watchdog is monitoring the selected channel. For Analog Watchdog 1: Only 1 channel can be monitored (or overall group of channels by setting parameter 'WatchdogMode'). For Analog Watchdog 2 and 3: Several channels can be monitored (by successive calls of '**HAL_ADC_AnalogWDGConfig()**' for each channel) This parameter can be a value of ***ADC_HAL_EC_AWD_NUMBER***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
  Configure the ADC analog watchdog mode: single/all/none channels. For Analog Watchdog 1: Configure the ADC analog watchdog mode: single channel or all channels, ADC groups regular and-or injected. For Analog Watchdog 2 and 3: Several channels can be monitored by applying successively the AWD init structure. For ADC1/ADC2, channels on ADC group regular and injected are not differentiated: Set value 'ADC_ANALOGWATCHDOG_SINGLE_xxx' to monitor 1 channel, value 'ADC_ANALOGWATCHDOG_ALL_xxx' to monitor all channels, value 'ADC_ANALOGWATCHDOG_NONE' to monitor no channel. This parameter can be a value of ***ADC_analog_watchdog_mode***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
  Select which ADC channel to monitor by analog watchdog. For Analog Watchdog 1: this parameter has an effect only if parameter 'WatchdogMode' is configured on single channel (only 1 channel can be monitored). For Analog Watchdog 2 and 3: Several channels can be monitored. To use this feature, call successively the function **HAL_ADC_AnalogWDGConfig()** for each channel to be added (or removed with value 'ADC_ANALOGWATCHDOG_NONE'). Note: 'ORING' channels is not supported. For Analog watchdog 2 and 3 and for multiple channels monitoring it's mandatory to recall successively the function **HAL_ADC_AnalogWDGConfig()** to add monitored channel for each call. This parameter can be a value of ***ADC_HAL_EC_CHANNEL***.
- ***FunctionalState ADC_AnalogWDGConfTypeDef::ITMode***
  Specify whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  Configure the ADC analog watchdog High threshold value. Depending of ADC resolution selected (14, 12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0x3FFF, 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits:
  - if ADC resolution is 12 bits the 4 LSB are ignored,
  - if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog Low threshold value. Depending of ADC resolution selected (14, 12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0x3FFF, 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: Analog watchdog 2 and 3 are limited to a resolution of 8 bits:
  - if ADC resolution is 12 bits the 4 LSB are ignored,
  - if ADC resolution is 10 bits the 2 LSB are ignored. Note: If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).
- ***uint32_t ADC_AnalogWDGConfTypeDef::FilteringConfig***
  Specify whether filtering should be use and the number of samples to consider. Before setting flag or raising interrupt, analog watchdog can wait to have several consecutive out-of-window samples. This parameter allows to configure this number. This parameter only applies to Analog watchdog 1. For others, use value ADC_AWD_FILTERING_NONE. Note: This parameter is applicable for ADC1/ADC2 on this device. This parameter can be a value of ***ADC_analog_watchdog_filtering_config***.

7.1.5 ADC_InjectionConfigTypeDef

ADC_InjectionConfigTypeDef is defined in the stm32u5xx_hal_adc.h

Data Fields

- ***uint32_t ContextQueue***
- ***uint32_t ChannelCount***

Field Documentation

- ***uint32_t ADC_InjectionConfigTypeDef::ContextQueue***
  Injected channel configuration context: build-up over each **HAL_ADCEx_InjectedConfigChannel()** call to finally initialize JSQR register at **HAL_ADCEx_InjectedConfigChannel()** last call
- ***uint32_t ADC_InjectionConfigTypeDef::ChannelCount***
  Number of channels in the injected sequence

7.1.6 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32u5xx_hal_adc.h

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***ADC_InjectionConfigTypeDef InjectionConfig***
- ***uint32_t ADCGroupRegularSequencerRanks***
- ***void(* ConvCpltCallback***
- ***void(* ConvHalfCpltCallback***
- ***void(* LevelOutOfWindowCallback***
- ***void(* ErrorCallback***
- ***void(* InjectedConvCpltCallback***
- ***void(* InjectedQueueOverflowCallback***
- ***void(* LevelOutOfWindow2Callback***
- ***void(* LevelOutOfWindow3Callback***
- ***void(* EndOfSamplingCallback***
- ***void(* CalibrationCpltCallback***
- ***void(* VoltageRegulatorCallback***
- ***void(* ADCReadyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***ADC_TypeDef* __ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef __ADC_HandleTypeDef::Init***
  ADC initialization parameters and regular conversions setting
- ***DMA_HandleTypeDef* __ADC_HandleTypeDef::DMA_Handle***
  Pointer DMA Handler
- ***HAL_LockTypeDef __ADC_HandleTypeDef::Lock***
  ADC locking object
- ***__IO uint32_t __ADC_HandleTypeDef::State***
  ADC communication state (bitmap of ADC states)
- ***__IO uint32_t __ADC_HandleTypeDef::ErrorCode***
  ADC Error code
- ***ADC_InjectionConfigTypeDef __ADC_HandleTypeDef::InjectionConfig***
  ADC injected channel configuration build-up structure
- ***uint32_t __ADC_HandleTypeDef::ADCGroupRegularSequencerRanks***
  ADC group regular sequencer memorization of ranks setting, used in mode "fully configurable" (refer to parameter 'ScanConvMode')
- ***void(* __ADC_HandleTypeDef::ConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion complete callback

- ***void(* __ADC_HandleTypeDef::ConvHalfCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC conversion DMA half-transfer callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 1 callback
- ***void(* __ADC_HandleTypeDef::ErrorCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC error callback
- ***void(* __ADC_HandleTypeDef::InjectedConvCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected conversion complete callback
- ***void(* __ADC_HandleTypeDef::InjectedQueueOverflowCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC group injected context queue overflow callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow2Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 2 callback
- ***void(* __ADC_HandleTypeDef::LevelOutOfWindow3Callback)(struct __ADC_HandleTypeDef *hadc)***
  ADC analog watchdog 3 callback
- ***void(* __ADC_HandleTypeDef::EndOfSamplingCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC end of sampling callback
- ***void(* __ADC_HandleTypeDef::CalibrationCpltCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC end of calibration callback
- ***void(* __ADC_HandleTypeDef::VoltageRegulatorCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC voltage regulator (LDO) Ready callback
- ***void(* __ADC_HandleTypeDef::ADCReadyCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Ready callback
- ***void(* __ADC_HandleTypeDef::MspInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp Init callback
- ***void(* __ADC_HandleTypeDef::MspDeInitCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Msp DeInit callback

7.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

7.2.1 ADC peripheral features

- 14-bit, 12-bit, 10-bit or 8-bit configurable resolution.
- Interrupt generation at the end of regular conversion and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.
- Programmable sampling time (channel wise)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- Configurable delay between conversions in Dual interleaved mode.
- ADC channels selectable single/differential input.
- ADC offset shared on 4 offset instances.
- ADC calibration
- ADC conversion of regular group.
- ADC supply requirements: 1.62 V to 3.6 V.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

7.2.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
   - As prerequisite, ADC clock must be configured at RCC top level.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Six possible clock sources: synchronous clock derived from AHB clock or asynchronous clock derived from system clock, the PLL2, the HSE, the HSI or the MSIK.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory) RCC_ADCCLKSOURCE_PLL2 enable: (optional: if asynchronous clock selected)
     - RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     - PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADCDAC;
     - PeriphClkInit.AdcDacClockSelection = RCC_ADCDACCLKSOURCE_PLL2;
     - HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
   - ADC clock source and clock prescaler are configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
   - Enable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_ENABLE()
   - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
   - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
4. Optionally, in case of usage of DMA:
   - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
   - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
   - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, group regular, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().

2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
   – ADC conversion by polling:
     ◦ Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
     ◦ Wait for ADC conversion completion using function HAL_ADC_PollForConversion()
     ◦ Retrieve conversion results using function HAL_ADC_GetValue()
     ◦ Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
   – ADC conversion by interruption:
     ◦ Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
     ◦ Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program)
     ◦ Retrieve conversion results using function HAL_ADC_GetValue()
     ◦ Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
   – ADC conversion with transfer by DMA:
     ◦ Activate the ADC peripheral and start conversions using function HAL_ADC_Start_DMA()
     ◦ Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() or HAL_ADC_ConvHalfCpltCallback() (these functions must be implemented in user program)
     ◦ Conversion results are automatically transferred by DMA into destination variable address.
     ◦ Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_DMA()

Note: Callback functions must be implemented in user program:
- *HAL_ADC_ErrorCallback()*
- *HAL_ADC_LevelOutOfWindowCallback() (callback of analog watchdog)*
- *HAL_ADC_ConvCpltCallback()*
- *HAL_ADC_ConvHalfCpltCallback*

Deinitialization of ADC

1. Disable the ADC interface
   – ADC clock can be hard reset and disabled at RCC top level.
   – Hard reset of ADC peripherals using macro __HAL_RCC_ADCx_FORCE_RESET(), __HAL_RCC_ADCx_RELEASE_RESET().
   – ADC clock disable using the equivalent macro/functions as configuration step.
     ◦ Example: Into HAL_ADC_MspDeInit() (recommended code location) or with other device clock parameters configuration:
     ◦ __HAL_RCC_ADC_CLK_DISABLE(); (if not used anymore) RCC_ADCDACCLKSOURCE_HSI restore: (optional)
     ◦ RCC_PeriphClkInitTypeDef RCC_PeriphClkInit;
     ◦ PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADCDAC;
     ◦ PeriphClkInit.AdcDacClockSelection = RCC_ADCDACCLKSOURCE_HSI;
     ◦ HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
2. ADC pins configuration
   – Disable the clock for the ADC GPIOs using macro __HAL_RCC_GPIOx_CLK_DISABLE()
3. Optionally, in case of usage of ADC with interruptions:
   – Disable the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)
4. Optionally, in case of usage of DMA:
   – Deinitialize the DMA using function HAL_DMA_Init().
   – Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

Callback registration

The compilation flag USE_HAL_ADC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_ADC_RegisterCallback() to register an interrupt callback.

Function @ref HAL_ADC_RegisterCallback() allows to register following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_ADC_UnRegisterCallback to reset a callback to the default weak function.

@ref HAL_ADC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ConvCpltCallback : ADC conversion complete callback
- ConvHalfCpltCallback : ADC conversion DMA half-transfer callback
- LevelOutOfWindowCallback : ADC analog watchdog 1 callback
- ErrorCallback : ADC error callback
- InjectedConvCpltCallback : ADC group injected conversion complete callback
- InjectedQueueOverflowCallback : ADC group injected context queue overflow callback
- LevelOutOfWindow2Callback : ADC analog watchdog 2 callback
- LevelOutOfWindow3Callback : ADC analog watchdog 3 callback
- EndOfSamplingCallback : ADC end of sampling callback
- MspInitCallback : ADC Msp Init callback
- MspDeInitCallback : ADC Msp DeInit callback

By default, after the @ref HAL_ADC_Init() and when the state is @ref HAL_ADC_STATE_RESET all callbacks are set to the corresponding weak functions: examples @ref HAL_ADC_ConvCpltCallback(), @ref HAL_ADC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_ADC_Init()/ @ref HAL_ADC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the @ref HAL_ADC_Init()/ @ref HAL_ADC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_ADC_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in @ref HAL_ADC_STATE_READY or @ref HAL_ADC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_ADC_RegisterCallback() before calling @ref HAL_ADC_DeInit() or @ref HAL_ADC_Init() function.

When the compilation flag USE_HAL_ADC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

7.2.3 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- *HAL_ADC_ConfigChannel()*
- *HAL_ADC_AnalogWDGConfig()*

7.2.4 Peripheral state and errors functions

This subsection provides functions to get in run-time the status of the peripheral.

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:
- *HAL_ADC_GetState()*
- *HAL_ADC_GetError()*

7.2.5 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- As prerequisite, ADC clock must be configured at RCC top level (refer to description of RCC configuration for ADC in header of this file).
- Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef".
- This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".
- Parameters related to common ADC registers (ADC clock mode) are set only if all ADCs are disabled. If this is not the case, these common parameters setting are bypassed without error reporting: it can be the intended behaviour in case of update of a parameter of ADC_InitTypeDef on the fly, without disabling the other ADCs.

HAL_ADC_DeInit

Function name

HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)

Function description

Deinitialize the ADC peripheral registers to their default reset values, with deinitialization of the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- For devices with several ADCs: reset of ADC common registers is done only if all ADCs sharing the same common group are disabled. (function "HAL_ADC_MspDeInit()" is also called under the same conditions: all ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances). If this is not the case, reset of these common parameters reset is bypassed without error reporting: it can be the intended behavior in case of reset of a single ADC while the other ADCs sharing the same common group is still running.
- By default, HAL_ADC_DeInit() set ADC in mode deep power-down: this saves more power by reducing leakage currents and is particularly interesting before entering MCU low-power modes.

HAL_ADC_MspInit

Function name

void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_MspDeInit

Function name

void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)

Function description

DeInitialize the ADC MSP.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- All ADC instances use the same core clock at RCC level, disabling the core clock reset all ADC instances).

HAL_ADC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_RegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID, pADC_CallbackTypeDef pCallback)

Function description

Register a User ADC Callback To be used instead of the weak predefined callback.

Parameters

- **hadc:** Pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_ADC_CONVERSION_COMPLETE_CB_ID ADC conversion complete callback ID
  - HAL_ADC_CONVERSION_HALF_CB_ID ADC conversion DMA half-transfer callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC analog watchdog 1 callback ID
  - HAL_ADC_ERROR_CB_ID ADC error callback ID
  - HAL_ADC_INJ_CONVERSION_COMPLETE_CB_ID ADC group injected conversion complete callback ID
  - HAL_ADC_INJ_QUEUE_OVEFLOW_CB_ID ADC group injected context queue overflow callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_2_CB_ID ADC analog watchdog 2 callback ID
  - HAL_ADC_LEVEL_OUT_OF_WINDOW_3_CB_ID ADC analog watchdog 3 callback ID
  - HAL_ADC_END_OF_SAMPLING_CB_ID ADC end of sampling callback ID
  - HAL_ADC_END_OF_CALIBRATION_CB_ID ADC end of calibration callback ID
  - HAL_ADC_VOLTAGE_REGULATOR_CB_ID ADC voltage regulator (LDO) Ready callback ID
  - HAL_ADC_ADC_READY_CB_ID ADC Ready callback ID
  - HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  - HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit callback ID
  - HAL_ADC_MSPINIT_CB_ID MspInit callback ID
  - HAL_ADC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_ADC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_ADC_UnRegisterCallback (ADC_HandleTypeDef * hadc, HAL_ADC_CallbackIDTypeDef CallbackID)

Function description

Unregister a ADC Callback ADC callback is redirected to the weak predefined callback.

Parameters

- **hadc:** Pointer to a ADC_HandleTypeDef structure that contains the configuration information for the specified ADC.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_ADC_CONVERSION_COMPLETE_CB_ID ADC conversion complete callback ID
  – HAL_ADC_CONVERSION_HALF_CB_ID ADC conversion DMA half-transfer callback ID
  – HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC analog watchdog 1 callback ID
  – HAL_ADC_ERROR_CB_ID ADC error callback ID
  – HAL_ADC_INJ_CONVERSION_COMPLETE_CB_ID ADC group injected conversion complete callback ID
  – HAL_ADC_INJ_QUEUE_OVEFLOW_CB_ID ADC group injected context queue overflow callback ID
  – HAL_ADC_LEVEL_OUT_OF_WINDOW_2_CB_ID ADC analog watchdog 2 callback ID
  – HAL_ADC_LEVEL_OUT_OF_WINDOW_3_CB_ID ADC analog watchdog 3 callback ID
  – HAL_ADC_END_OF_SAMPLING_CB_ID ADC end of sampling callback ID
  – HAL_ADC_END_OF_CALIBRATION_CB_ID ADC end of calibration callback ID
  – HAL_ADC_VOLTAGE_REGULATOR_CB_ID ADC voltage regulator (LDO) Ready callback ID
  – HAL_ADC_ADC_READY_CB_ID ADC Ready callback ID
  – HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  – HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit callback ID
  – HAL_ADC_MSPINIT_CB_ID MspInit callback ID
  – HAL_ADC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_ADC_Start

Function name

HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled (when multimode feature is available): if ADC is Slave, ADC is enabled but conversion is not started, if ADC is master, ADC is enabled and multimode conversion is started.

HAL_ADC_Stop

Function name

HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for regular group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue().
- This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

HAL_ADC_PollForEvent

Function name

HAL_StatusTypeDef HAL_ADC_PollForEvent (ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function description

Poll for ADC event.

Parameters

- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values:
  - ADC_EOSMP_EVENT ADC End of Sampling event
  - ADC_AWD1_EVENT ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 devices)
  - ADC_AWD2_EVENT ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 families)
  - ADC_AWD3_EVENT ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 families)
  - ADC_OVR_EVENT ADC Overrun event
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- The relevant flag is cleared if found to be set, except for ADC_FLAG_OVR. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_OVR_DATA_OVERWRITTEN. Otherwise, data register may be potentially overwritten by a new converted data as soon as OVR is cleared. To reset OVR flag once the preserved data is retrieved, the user can resort to macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_OVR);

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of regular group with interruption.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function according to initialization setting : EOC (end of conversion), EOS (end of sequence), OVR overrun. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_IT() must be called for ADC Slave first, then for ADC Master. For ADC Slave, ADC is enabled only (conversion is not started). For ADC Master, ADC is enabled and multimode conversion is started.
- To guarantee a proper reset of all interruptions once all the needed conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure a correct stop of the IT-based conversions.
- By default, HAL_ADC_Start_IT() does not enable the End Of Sampling interruption. If required (e.g. in case of oversampling with trigger mode), the user must: 1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSMP) 2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP) before calling HAL_ADC_Start_IT().

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interrution of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADC_Start_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, const uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start conversion of regular group and transfer result through DMA.

Parameters

- **hadc:** ADC handle
- **pData:** Destination Buffer address.
- **Length:** Number of data to be transferred from ADC peripheral to memory

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function: overrun (if applicable), DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA() is designed for single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStart_DMA() function must be used.

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion on ADC group injected. If ADC group injected is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.
- Case of multimode enabled (when multimode feature is available): HAL_ADC_Stop_DMA() function is dedicated to single-ADC mode only. For multimode, the dedicated HAL_ADCEx_MultiModeStop_DMA() API must be used.

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (const ADC_HandleTypeDef * hadc)

Function description

Get ADC regular group conversion result.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** group regular conversion data

Notes

- Reading register DR automatically clears ADC flag EOC (ADC group regular end of unitary conversion).
- This function does not clear ADC flag EOS (ADC group regular end of sequence conversion). Occurrence of flag EOS rising: If sequencer is composed of 1 rank, flag EOS is equivalent to flag EOC.If sequencer is composed of several ranks, during the scan sequence flag EOC only is raised, at the end of the scan sequence both flags EOC and EOS are raised. To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).

HAL_ADC_IRQHandler

Function name

void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)

Function description

Handle ADC interrupt request.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvCpltCallback

Function name

void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConvHalfCpltCallback

Function name

void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Conversion DMA half-transfer callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_LevelOutOfWindowCallback

Function name

void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 1 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ErrorCallback

Function name

void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)

Function description

ADC error callback in non-blocking mode (ADC conversion with interruption or transfer by DMA).

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- In case of error due to overrun when using ADC with DMA transfer (HAL ADC handle parameter "ErrorCode" to state "HAL_ADC_ERROR_OVR"): Reinitialize the DMA using function "HAL_ADC_Stop_DMA()".If needed, restart a new ADC conversion using function "HAL_ADC_Start_DMA()" (this function is also clearing overrun flag)

HAL_ADC_CalibrationCpltCallback

Function name

void HAL_ADC_CalibrationCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Calibration complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_VoltageRegulatorCallback

Function name

void HAL_ADC_VoltageRegulatorCallback (ADC_HandleTypeDef * hadc)

Function description

Voltage Regulator (LDO) Ready callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ADCReadyCallback

Function name

void HAL_ADC_ADCReadyCallback (ADC_HandleTypeDef * hadc)

Function description

ADC Ready callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * pConfig)

Function description

Configure a channel to be assigned to ADC group regular.

Parameters

- **hadc:** ADC handle
- **pConfig:** Structure of ADC channel assigned to ADC group regular.

Return values

- **HAL:** status

Notes

- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Possibility to update parameters on the fly: This function initializes channel into ADC group regular, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_ChannelConfTypeDef".

HAL_ADC_AnalogWDGConfig

Function name

HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * pAnalogWDGConfig)

Function description

Configure the analog watchdog.

Parameters

- **hadc:** ADC handle
- **pAnalogWDGConfig:** Structure of ADC analog watchdog configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes the selected analog watchdog, successive calls to this function can be used to reconfigure some parameters of structure "ADC_AnalogWDGConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_AnalogWDGConfTypeDef".

HAL_ADC_GetState

Function name

uint32_t HAL_ADC_GetState (const ADC_HandleTypeDef * hadc)

Function description

Return the ADC handle state.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** handle state (bitfield on 32 bits)

Notes

- ADC state machine is managed by bitfields, ADC status must be compared with states bits. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) "

HAL_ADC_GetError

Function name

uint32_t HAL_ADC_GetError (const ADC_HandleTypeDef * hadc)

Function description

Return the ADC error code.

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** error code (bitfield on 32 bits)

ADC_ConversionStop

Function name

HAL_StatusTypeDef ADC_ConversionStop (ADC_HandleTypeDef * hadc, uint32_t ConversionGroup)

Function description

Stop ADC conversion.

Parameters

- **hadc:** ADC handle
- **ConversionGroup:** ADC group regular and/or injected. This parameter can be one of the following values:
  – ADC_REGULAR_GROUP ADC regular conversion type.
  – ADC_INJECTED_GROUP ADC injected conversion type.
  – ADC_REGULAR_INJECTED_GROUP ADC regular and injected conversion type.

Return values

- **HAL:** status.

ADC_Enable

Function name

HAL_StatusTypeDef ADC_Enable (ADC_HandleTypeDef * hadc)

Function description

Enable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC must be disabled and voltage regulator must be enabled (done into HAL_ADC_Init()).

ADC_Disable

Function name

HAL_StatusTypeDef ADC_Disable (ADC_HandleTypeDef * hadc)

Function description

Disable the selected ADC.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- Prerequisite condition to use this function: ADC conversions must be stopped.

ADC_DMAConvCplt

Function name

void ADC_DMAConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAHalfConvCplt

Function name

void ADC_DMAHalfConvCplt (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_DMAError

Function name

void ADC_DMAError (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

ADC_ConfigureBoostMode

Function name

void ADC_ConfigureBoostMode (ADC_HandleTypeDef * hadc)

Function description

7.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

7.3.1 ADC

ADC

ADC Analog Watchdog filtering configuration

ADC_AWD_FILTERING_NONE

ADC analog watchdog no filtering, one out-of-window sample is needed to raise flag or interrupt

ADC_AWD_FILTERING_2SAMPLES

ADC analog watchdog 2 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_3SAMPLES

ADC analog watchdog 3 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_4SAMPLES

ADC analog watchdog 4 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_5SAMPLES

ADC analog watchdog 5 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_6SAMPLES

ADC analog watchdog 6 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_7SAMPLES

ADC analog watchdog 7 consecutives out-of-window samples are needed to raise flag or interrupt

ADC_AWD_FILTERING_8SAMPLES

ADC analog watchdog 8 consecutives out-of-window samples are needed to raise flag or interrupt

ADC Analog Watchdog Mode

ADC_ANALOGWATCHDOG_NONE

No analog watchdog selected

ADC_ANALOGWATCHDOG_SINGLE_REG

Analog watchdog applied to a regular group single channel

ADC_ANALOGWATCHDOG_SINGLE_INJEC

Analog watchdog applied to an injected group single channel

ADC_ANALOGWATCHDOG_SINGLE_REGINJEC

Analog watchdog applied to a regular and injected groups single channel

ADC_ANALOGWATCHDOG_ALL_REG

Analog watchdog applied to regular group all channels

ADC_ANALOGWATCHDOG_ALL_INJEC

Analog watchdog applied to injected group all channels

ADC_ANALOGWATCHDOG_ALL_REGINJEC

Analog watchdog applied to regular and injected groups all channels

ADCx CFGR fields

ADC_CFGR_FIELDS

ADCx CFGR sub fields

ADC_CFGR_FIELDS_2

ADC Conversion Data Management

ADC_CONVERSIONDATA_DR

Regular Conversion data stored in DR register only

ADC_CONVERSIONDATA_MDF

MDF mode selected

ADC_CONVERSIONDATA_DMA_ONESHOT

DMA one shot mode selected

ADC_CONVERSIONDATA_DMA_CIRCULAR

DMA circular mode selected

ADC sequencer end of unitary conversion or sequence conversions

ADC_EOC_SINGLE_CONV

End of unitary conversion flag

ADC_EOC_SEQ_CONV

End of sequence conversions flag

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

ADC peripheral internal error (problem of clocking, enable/disable, erroneous state, ...)

HAL_ADC_ERROR_OVR

Overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_INVALID_CALLBACK

Invalid Callback error

ADC Event type

ADC_EOSMP_EVENT

ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog, present on all STM32 series)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog, not present on all STM32 series)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog, not present on all STM32 series)

ADC_OVR_EVENT

ADC overrun event

ADC Exported Constants

ADC_AWD_EVENT

ADC Analog watchdog 1 event: Naming for compatibility with other STM32 devices having only one analog watchdog

ADC flags definition

ADC_FLAG_RDY

ADC Ready flag

ADC_FLAG_EOSMP

ADC End of Sampling flag

ADC_FLAG_EOC

ADC End of Regular Conversion flag

ADC_FLAG_EOS

ADC End of Regular sequence of Conversions flag

ADC_FLAG_OVR

ADC overrun flag

ADC_FLAG_JEOC

ADC End of Injected Conversion flag

ADC_FLAG_JEOS

ADC End of Injected sequence of Conversions flag

ADC_FLAG_AWD1

ADC Analog watchdog 1 flag (main analog watchdog)

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag (additional analog watchdog)

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag (additional analog watchdog)

ADC_FLAG_EOCAL

ADC End of Calibration flag

ADC_FLAG_LDORDY

ADC Voltage Regulator (LDO) Ready flag

Analog watchdog - Analog watchdog number

ADC_ANALOGWATCHDOG_1

ADC analog watchdog number 1

ADC_ANALOGWATCHDOG_2

ADC analog watchdog number 2

ADC_ANALOGWATCHDOG_3

ADC analog watchdog number 3

ADC instance - Channel number

ADC_CHANNEL_0

ADC external channel (channel connected to GPIO pin) ADCx_IN0

ADC_CHANNEL_1

ADC external channel (channel connected to GPIO pin) ADCx_IN1

ADC_CHANNEL_2

ADC external channel (channel connected to GPIO pin) ADCx_IN2

ADC_CHANNEL_3

ADC external channel (channel connected to GPIO pin) ADCx_IN3

ADC_CHANNEL_4

ADC external channel (channel connected to GPIO pin) ADCx_IN4

ADC_CHANNEL_5

ADC external channel (channel connected to GPIO pin) ADCx_IN5

ADC_CHANNEL_6

ADC external channel (channel connected to GPIO pin) ADCx_IN6

ADC_CHANNEL_7

ADC external channel (channel connected to GPIO pin) ADCx_IN7

ADC_CHANNEL_8

ADC external channel (channel connected to GPIO pin) ADCx_IN8

ADC_CHANNEL_9

ADC external channel (channel connected to GPIO pin) ADCx_IN9

ADC_CHANNEL_10

ADC external channel (channel connected to GPIO pin) ADCx_IN10

ADC_CHANNEL_11

ADC external channel (channel connected to GPIO pin) ADCx_IN11

ADC_CHANNEL_12

ADC external channel (channel connected to GPIO pin) ADCx_IN12

ADC_CHANNEL_13

ADC external channel (channel connected to GPIO pin) ADCx_IN13

ADC_CHANNEL_14

ADC external channel (channel connected to GPIO pin) ADCx_IN14

ADC_CHANNEL_15

ADC external channel (channel connected to GPIO pin) ADCx_IN15

ADC_CHANNEL_16

ADC external channel (channel connected to GPIO pin) ADCx_IN16

ADC_CHANNEL_17

ADC external channel (channel connected to GPIO pin) ADCx_IN17

ADC_CHANNEL_18

ADC external channel (channel connected to GPIO pin) ADCx_IN18

ADC_CHANNEL_19

ADC external channel (channel connected to GPIO pin) ADCx_IN19

ADC_CHANNEL_20

ADC external channel (channel connected to GPIO pin) ADCx_IN20

ADC_CHANNEL_21

ADC external channel (channel connected to GPIO pin) ADCx_IN21

ADC_CHANNEL_22

ADC external channel (channel connected to GPIO pin) ADCx_IN22

ADC_CHANNEL_23

ADC external channel (channel connected to GPIO pin) ADCx_IN23

ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference.

ADC_CHANNEL_TEMPSENSOR

ADC internal channel connected to Temperature sensor.

ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda.

ADC_CHANNEL_DAC1CH1_ADC4

ADC internal channel connected to DAC1 channel 1, channel specific to ADC4

ADC_CHANNEL_DAC1CH2_ADC4

ADC internal channel connected to DAC1 channel 2, channel specific to ADC4

ADC4_CHANNEL_TEMPSENSOR

ADC internal channel connected to Temperature sensor, channel specific to ADC4.

ADC4_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, channel specific to ADC4.

ADC_CHANNEL_VCORE

ADC internal channel connected to Vcore, channel specific to ADC4.

Channel - Sampling time

ADC_SAMPLETIME_5CYCLES

Sampling time 5 ADC clock cycles

ADC_SAMPLETIME_6CYCLES

Sampling time 6 ADC clock cycles

ADC_SAMPLETIME_12CYCLES

Sampling time 12 ADC clock cycles

ADC_SAMPLETIME_20CYCLES

Sampling time 20 ADC clock cycles

ADC_SAMPLETIME_36CYCLES

Sampling time 36 ADC clock cycles

ADC_SAMPLETIME_68CYCLES

Sampling time 68 ADC clock cycles

ADC_SAMPLETIME_391CYCLES

Sampling time 391 ADC clock cycles

ADC_SAMPLETIME_814CYCLES

Sampling time 814 ADC clock cycles

ADC4_SAMPLETIME_1CYCLE_5

Sampling time 1.5 ADC clock cycle

ADC4_SAMPLETIME_3CYCLES_5

Sampling time 3.5 ADC clock cycles

ADC4_SAMPLETIME_7CYCLES_5

Sampling time 7.5 ADC clock cycles

ADC4_SAMPLETIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

ADC4_SAMPLETIME_19CYCLES_5

Sampling time 19.5 ADC clock cycles

ADC4_SAMPLETIME_39CYCLES_5

Sampling time 39.5 ADC clock cycles

ADC4_SAMPLETIME_79CYCLES_5

Sampling time 79.5 ADC clock cycles

ADC4_SAMPLETIME_814CYCLES_5

Sampling time 814.5 ADC clock cycles

Channel - Single or differential ending

ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

ADC common - Clock source

ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC conversion data alignment

ADC_DATAALIGN_RIGHT

ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)

ADC_DATAALIGN_LEFT

ADC conversion data alignment: left aligned (alignment on data register MSB bit 15)

ADC instance - Groups

ADC_REGULAR_GROUP

ADC group regular (available on all STM32 devices)

ADC_INJECTED_GROUP

ADC group injected (not available on all STM32 devices)

ADC_REGULAR_INJECTED_GROUP

ADC both groups regular and injected

Multimode - DMA transfer mode depending on ADC resolution

ADC_DMAACCESSMODE_DISABLED

DMA multimode disabled: each ADC uses its own DMA channel

ADC_DMAACCESSMODE_12_10_BITS

DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 12 and 10 bits resolution

ADC_DMAACCESSMODE_8_6_BITS

DMA multimode enabled (one DMA channel for both ADC, DMA of ADC master) for 8 and 6 bits resolution

Multimode - Mode

ADC_MODE_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

ADC_DUALMODE_REGSIMULT

ADC dual mode enabled: group regular simultaneous

ADC_DUALMODE_INTERL

ADC dual mode enabled: Combined group regular interleaved

ADC_DUALMODE_INJECSIMULT

ADC dual mode enabled: group injected simultaneous

ADC_DUALMODE_ALTERTRIG

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

ADC_DUALMODE_REGSIMULT_INJECSIMULT

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

ADC_DUALMODE_REGSIMULT_ALTERTRIG

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

ADC_DUALMODE_REGINTERL_INJECSIMULT

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

ADC_TWOSAMPLINGDELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle

ADC_TWOSAMPLINGDELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles

ADC_TWOSAMPLINGDELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles

ADC_TWOSAMPLINGDELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles

ADC_TWOSAMPLINGDELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles

ADC_TWOSAMPLINGDELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles

ADC_TWOSAMPLINGDELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles

ADC_TWOSAMPLINGDELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles

ADC_TWOSAMPLINGDELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles

ADC_TWOSAMPLINGDELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles

ADC_TWOSAMPLINGDELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles

ADC_TWOSAMPLINGDELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles

ADC_TWOSAMPLINGDELAY_13CYCLES

ADC multimode delay between two sampling phases: 13 ADC clock cycles

ADC_TWOSAMPLINGDELAY_14CYCLES

ADC multimode delay between two sampling phases: 14 ADC clock cycles

ADC_TWOSAMPLINGDELAY_15CYCLES

ADC multimode delay between two sampling phases: 15 ADC clock cycles

ADC_TWOSAMPLINGDELAY_16CYCLES

ADC multimode delay between two sampling phases: 16 ADC clock cycles

ADC instance - Offset number

ADC_OFFSET_NONE

ADC offset disabled: no offset correction for the selected ADC channel

ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Oversampling - Discontinuous mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

ADC_TRIGGEREDMODE_MULTI_TRIGGER

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

ADC_OVERSAMPLING_RATIO_2

ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_4

ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_8

ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_16

ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_32

ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_64

ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_128

ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

ADC_OVERSAMPLING_RATIO_256

ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

Oversampling - Oversampling scope for ADC group regular

ADC_REGOVERSAMPLING_CONTINUED_MODE

Oversampling buffer maintained during injection sequence

ADC_REGOVERSAMPLING_RESUMED_MODE

Oversampling buffer zeroed during injection sequence

Oversampling - Data shift

ADC_RIGHTBITSHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_1

ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_2

ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_3

ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_4

ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_5

ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_6

ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_7

ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_8

ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_9

ADC oversampling shift of 9 (sum of the ADC conversions data is divided by 512 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_10

ADC oversampling shift of 10 (sum of the ADC conversions data is divided by 1024 to result as the ADC oversampling conversion data)

ADC_RIGHTBITSHIFT_11

ADC oversampling shift of 11 (sum of the ADC conversions data is divided by 2048 to result as the ADC oversampling conversion data)

ADC group regular - Overrun behavior on conversion data

ADC_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

ADC_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sequencer ranks

ADC_REGULAR_RANK_1

ADC group regular sequencer rank 1

ADC_REGULAR_RANK_2

ADC group regular sequencer rank 2

ADC_REGULAR_RANK_3

ADC group regular sequencer rank 3

ADC_REGULAR_RANK_4

ADC group regular sequencer rank 4

ADC_REGULAR_RANK_5

ADC group regular sequencer rank 5

ADC_REGULAR_RANK_6

ADC group regular sequencer rank 6

ADC_REGULAR_RANK_7

ADC group regular sequencer rank 7

ADC_REGULAR_RANK_8

ADC group regular sequencer rank 8

ADC_REGULAR_RANK_9

ADC group regular sequencer rank 9

ADC_REGULAR_RANK_10

ADC group regular sequencer rank 10

ADC_REGULAR_RANK_11

ADC group regular sequencer rank 11

ADC_REGULAR_RANK_12

ADC group regular sequencer rank 12

ADC_REGULAR_RANK_13

ADC group regular sequencer rank 13

ADC_REGULAR_RANK_14

ADC group regular sequencer rank 14

ADC_REGULAR_RANK_15

ADC group regular sequencer rank 15

ADC_REGULAR_RANK_16

ADC group regular sequencer rank 16

ADC4_RANK_CHANNEL_NUMBER

Setting relevant if parameter "ScanConvMode" is set to sequencer not fully configurable: Enable the rank of the selected channels. Number of ranks in the sequence is defined by number of channels enabled, rank of each channel is defined by channel number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...)

ADC4_RANK_NONE

Setting relevant if parameter "ScanConvMode" is set to sequencer not fully configurable: Disable the selected rank (selected channel) from sequencer

ADC4_REGULAR_RANK_1

ADC group regular sequencer rank 1

ADC4_REGULAR_RANK_2

ADC group regular sequencer rank 2

ADC4_REGULAR_RANK_3

ADC group regular sequencer rank 3

ADC4_REGULAR_RANK_4

ADC group regular sequencer rank 4

ADC4_REGULAR_RANK_5

ADC group regular sequencer rank 5

ADC4_REGULAR_RANK_6

ADC group regular sequencer rank 6

ADC4_REGULAR_RANK_7

ADC group regular sequencer rank 7

ADC4_REGULAR_RANK_8

ADC group regular sequencer rank 8

ADC group regular - Trigger frequency mode

ADC_TRIGGER_FREQ_HIGH

ADC trigger frequency mode set to high frequency. Note: ADC trigger frequency mode must be set to low frequency when a duration is exceeded before ADC conversion start trigger event (between ADC enable and ADC conversion start trigger event or between two ADC conversion start trigger event). Duration value: Refer to device datasheet, parameter "tIdle".

ADC_TRIGGER_FREQ_LOW

ADC trigger frequency mode set to low frequency. Note: ADC trigger frequency mode must be set to low frequency when a duration is exceeded before ADC conversion start trigger event (between ADC enable and ADC conversion start trigger event or between two ADC conversion start trigger event). Duration value: Refer to device datasheet, parameter "tIdle".

ADC instance - Resolution

ADC_RESOLUTION_14B

ADC resolution 14 bits (ADC1, ADC2 only)

ADC_RESOLUTION_12B

ADC resolution 12 bits

ADC_RESOLUTION_10B

ADC resolution 10 bits

ADC_RESOLUTION_8B

ADC resolution 8 bits

ADC_RESOLUTION_6B

ADC resolution 6 bits (ADC4 only)

ADC4_RESOLUTION_12B

ADC4_RESOLUTION_10B

ADC4_RESOLUTION_8B

ADC4_RESOLUTION_6B

ADC instance - Sampling time common to a group of channels

ADC4_SAMPLINGTIME_COMMON_1

Set sampling time common to a group of channels: sampling time nb 1

ADC4_SAMPLINGTIME_COMMON_2

Set sampling time common to a group of channels: sampling time nb 2

HAL ADC macro to manage HAL ADC handle, IT and flags.

__HAL_ADC_RESET_HANDLE_STATE

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  - ADC_IT_RDY ADC Ready interrupt source
  - ADC_IT_EOSMP ADC End of Sampling interrupt source
  - ADC_IT_EOC ADC End of Regular Conversion interrupt source
  - ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  - ADC_IT_OVR ADC overrun interrupt source
  - ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  - ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  - ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  - ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  - ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  - ADC_IT_EOCAL ADC End of Calibration interrupt source
  - ADC_IT_LDORDY ADC Voltage Regulator (LDO) Ready interrupt source

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be one of the following values:
  – ADC_IT_RDY ADC Ready interrupt source
  – ADC_IT_EOSMP ADC End of Sampling interrupt source
  – ADC_IT_EOC ADC End of Regular Conversion interrupt source
  – ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  – ADC_IT_OVR ADC overrun interrupt source
  – ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  – ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  – ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  – ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  – ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  – ADC_IT_EOCAL ADC End of Calibration interrupt source
  – ADC_IT_LDORDY ADC Voltage Regulator (LDO) Ready interrupt source

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  – ADC_IT_RDY ADC Ready interrupt source
  – ADC_IT_EOSMP ADC End of Sampling interrupt source
  – ADC_IT_EOC ADC End of Regular Conversion interrupt source
  – ADC_IT_EOS ADC End of Regular sequence of Conversions interrupt source
  – ADC_IT_OVR ADC overrun interrupt source
  – ADC_IT_JEOC ADC End of Injected Conversion interrupt source
  – ADC_IT_JEOS ADC End of Injected sequence of Conversions interrupt source
  – ADC_IT_AWD1 ADC Analog watchdog 1 interrupt source (main analog watchdog)
  – ADC_IT_AWD2 ADC Analog watchdog 2 interrupt source (additional analog watchdog)
  – ADC_IT_AWD3 ADC Analog watchdog 3 interrupt source (additional analog watchdog)
  – ADC_IT_EOCAL ADC End of Calibration interrupt source
  – ADC_IT_LDORDY ADC Voltage Regulator (LDO) Ready interrupt source

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

- Check whether the specified ADC flag is set or not.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_EOCAL ADC End of Calibration flag
  - ADC_FLAG_LDORDY ADC Voltage Regulator (LDO) Ready flag

Return value:

- State: of flag (TRUE or FALSE).

__HAL_ADC_CLEAR_FLAG

Description:

- Clear the specified ADC flag.

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_FLAG_RDY ADC Ready flag
  - ADC_FLAG_EOSMP ADC End of Sampling flag
  - ADC_FLAG_EOC ADC End of Regular Conversion flag
  - ADC_FLAG_EOS ADC End of Regular sequence of Conversions flag
  - ADC_FLAG_OVR ADC overrun flag
  - ADC_FLAG_JEOC ADC End of Injected Conversion flag
  - ADC_FLAG_JEOS ADC End of Injected sequence of Conversions flag
  - ADC_FLAG_AWD1 ADC Analog watchdog 1 flag (main analog watchdog)
  - ADC_FLAG_AWD2 ADC Analog watchdog 2 flag (additional analog watchdog)
  - ADC_FLAG_AWD3 ADC Analog watchdog 3 flag (additional analog watchdog)
  - ADC_FLAG_EOCAL ADC End of Calibration flag
  - ADC_FLAG_LDORDY ADC Voltage Regulator (LDO) Ready flag

Return value:

- None

HAL ADC helper macro

__HAL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - ADC_CHANNEL_DAC1CH2_ADC4 (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __HAL_ADC_CHANNEL_TO_DECIMAL_NB(ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__HAL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  – ADC_CHANNEL_0 (3)
  – ADC_CHANNEL_1 (3)
  – ADC_CHANNEL_2 (3)
  – ADC_CHANNEL_3 (3)
  – ADC_CHANNEL_4 (3)
  – ADC_CHANNEL_5 (3)
  – ADC_CHANNEL_6
  – ADC_CHANNEL_7
  – ADC_CHANNEL_8
  – ADC_CHANNEL_9
  – ADC_CHANNEL_10
  – ADC_CHANNEL_11
  – ADC_CHANNEL_12
  – ADC_CHANNEL_13
  – ADC_CHANNEL_14
  – ADC_CHANNEL_15
  – ADC_CHANNEL_16
  – ADC_CHANNEL_17
  – ADC_CHANNEL_18
  – ADC_CHANNEL_VREFINT (1)
  – ADC_CHANNEL_TEMPSENSOR (1)
  – ADC_CHANNEL_VBAT (1)
  – ADC_CHANNEL_DAC1CH1_ADC4 (2)
  – ADC_CHANNEL_DAC1CH2_ADC4 (2)
  – ADC4_CHANNEL_TEMPSENSOR (2)
  – ADC4_CHANNEL_VBAT (2)
  – ADC_CHANNEL_VCORE (2)

Notes:

- Example: __HAL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "ADC_CHANNEL_4".

__HAL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - ADC4_CHANNEL_TEMPSENSOR (2)
  - ADC4_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VCORE (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): ADC_CHANNEL_1, ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_0 (3)
  - ADC_CHANNEL_1 (3)
  - ADC_CHANNEL_2 (3)
  - ADC_CHANNEL_3 (3)
  - ADC_CHANNEL_4 (3)
  - ADC_CHANNEL_5 (3)
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - ADC4_CHANNEL_TEMPSENSOR (2)
  - ADC4_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VCORE (2)

Return value:

- Returned: value can be one of the following values:
  - ADC_CHANNEL_0
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8
  - ADC_CHANNEL_9
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15
  - ADC_CHANNEL_16
  - ADC_CHANNEL_17
  - ADC_CHANNEL_18

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__HAL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - ADC_CHANNEL_VREFINT (1)
  - ADC_CHANNEL_TEMPSENSOR (1)
  - ADC_CHANNEL_VBAT (1)
  - ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - ADC4_CHANNEL_TEMPSENSOR (2)
  - ADC4_CHANNEL_VBAT (2)
  - ADC_CHANNEL_VCORE (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (ADC_CHANNEL_VREFINT, ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (ADC_CHANNEL_1, ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__HAL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  – LL_ADC_MULTI_MASTER
  – LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__HAL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__HAL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__HAL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__HAL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __ADCx__: ADC instance
- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of to the data to be converted This parameter can be one of the following values:
  - ADC_RESOLUTION_14B (1)
  - ADC_RESOLUTION_12B (1)(2)
  - ADC_RESOLUTION_10B (1)(2)
  - ADC_RESOLUTION_8B (1)(2)
  - ADC_RESOLUTION_6B (2)
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_14B (1)
  - ADC_RESOLUTION_12B (1)(2)
  - ADC_RESOLUTION_10B (1)(2)
  - ADC_RESOLUTION_8B (1)(2)
  - ADC_RESOLUTION_6B (2)

Return value:

- ADC: conversion data to the requested resolution

__HAL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __ADCx__: ADC instance
- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_14B (1)
  - ADC_RESOLUTION_12B (1)(2)
  - ADC_RESOLUTION_10B (1)(2)
  - ADC_RESOLUTION_8B (1)(2)
  - ADC_RESOLUTION_6B (2)

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__HAL_ADC_CALC_DIFF_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value) in differential ended mode.

Parameters:

- __ADCx__: ADC instance
- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - ADC_RESOLUTION_14B
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- On STM32U5, this feature is available on ADC instances: ADC1, ADC2. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__HAL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __ADCx__: ADC instance
- __VREFINT_ADC_DATA__: ADC conversion data (resolution 12 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  – ADC_RESOLUTION_14B (1)
  – ADC_RESOLUTION_12B (1)(2)
  – ADC_RESOLUTION_10B (1)(2)
  – ADC_RESOLUTION_8B (1)(2)
  – ADC_RESOLUTION_6B (2)

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 series, calibration data of internal voltage reference VrefInt corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __ADCx__: ADC instance
- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_14B (1)
  - ADC_RESOLUTION_12B (1)(2)
  - ADC_RESOLUTION_10B (1)(2)
  - ADC_RESOLUTION_8B (1)(2)
  - ADC_RESOLUTION_6B (2)

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 series, calibration data of temperature sensor corresponds to a resolution of 12 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 12 bits.

__HAL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __ADCx__: ADC instance
- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32U5, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). Refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage (see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - ADC_RESOLUTION_14B (1)
  - ADC_RESOLUTION_12B (1)(2)
  - ADC_RESOLUTION_10B (1)(2)
  - ADC_RESOLUTION_8B (1)(2)
  - ADC_RESOLUTION_6B (2)

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 12bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 bits.

ADC low power and deep power down selection

ADC_LOW_POWER_NONE

Both Low Power Auto Off and Deep Power Down is disabled

ADC_LOW_POWER_AUTOFF

Low Power Auto Off enabled and Deep Power Down is disabled

ADC_LOW_POWER_DPD

Low Power Auto Off disabled and Deep Power Down is enabled

ADC_LOW_POWER_AUTOFF_DPD

Low Power Auto Off and Deep Power Down are enabled

ADC VREF+ protection mode selection

ADC_VREF_PPROT_NONE

No VREF protection is applied

ADC_VREF_PPROT_VREFPROT

VREF+ protection when multiple ADCs are working simultaneously and a clock divider is used.

ADC_VREF_PPROT_VREFSECSMP

VREF+ protection when multiple ADCs are working simultaneously and a clock divider of 1 is used.

ADC_VREF_PPROT_VREF_VREFSECSMP

Both VREF+ protection when multiple ADCs are working simultaneously and VREF+ second sample protection.

ADC group injected trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGINJECCONV_EDGE_NONE

Injected conversions hardware trigger detection disabled

ADC_EXTERNALTRIGINJECCONV_EDGE_RISING

Injected conversions hardware trigger detection on the rising edge

ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING

Injected conversions hardware trigger detection on the falling edge

ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING

Injected conversions hardware trigger detection on both the rising and falling edges

ADC group injected trigger source

ADC_INJECTED_SOFTWARE_START

Software triggers injected group conversion start

ADC_EXTERNALTRIGINJEC_T1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_CC4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T2_CC1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_EXT_IT15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_CC4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T3_CC1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_T15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM1_CH2

ADC group injected conversion trigger from external peripheral: LPTIM1 CH2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM2_CH2

ADC group injected conversion trigger from external peripheral: LPTIM2 CH2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM3_CH1

ADC group injected conversion trigger from external peripheral: LPTIM3 CH1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIGINJEC_LPTIM4_OUT

ADC group injected conversion trigger from external peripheral: LPTIM4 OUT event. Trigger edge set to rising edge (default setting).

ADC group injected - Sequencer ranks

ADC_INJECTED_RANK_1

ADC group injected sequencer rank 1

ADC_INJECTED_RANK_2

ADC group injected sequencer rank 2

ADC_INJECTED_RANK_3

ADC group injected sequencer rank 3

ADC_INJECTED_RANK_4

ADC group injected sequencer rank 4

ADC interrupts definition

ADC_IT_RDY

ADC Ready interrupt source

ADC_IT_EOSMP

ADC End of sampling interrupt source

ADC_IT_EOC

ADC End of regular conversion interrupt source

ADC_IT_EOS

ADC End of regular sequence of conversions interrupt source

ADC_IT_OVR

ADC overrun interrupt source

ADC_IT_JEOC

ADC End of injected conversion interrupt source

ADC_IT_JEOS

ADC End of injected sequence of conversions interrupt source

ADC_IT_AWD1

ADC Analog watchdog 1 interrupt source (main analog watchdog)

ADC_IT_AWD2

ADC Analog watchdog 2 interrupt source (additional analog watchdog)

ADC_IT_AWD3

ADC Analog watchdog 3 interrupt source (additional analog watchdog)

ADC_IT_EOCAL

ADC End of Calibration interrupt source

ADC_IT_LDORDY

ADC Voltage Regulator (LDO) Ready interrupt source

ADC_IT_AWD

ADC Analog watchdog 1 interrupt source: naming for compatibility with other STM32 devices having only one analog watchdog

ADC group regular trigger edge (when external trigger is selected)

ADC_EXTERNALTRIGCONVEDGE_NONE

Regular conversions hardware trigger detection disabled

ADC_EXTERNALTRIGCONVEDGE_RISING

ADC group regular conversion trigger polarity set to rising edge

ADC_EXTERNALTRIGCONVEDGE_FALLING

ADC group regular conversion trigger polarity set to falling edge

ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular trigger source

ADC_SOFTWARE_START

ADC group regular conversion trigger internal: SW start.

ADC_EXTERNALTRIG_T1_CC1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_CC3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_CC2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_CC4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_EXT_IT11

ADC group regular conversion trigger from external peripheral: external interrupt line 11 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_T3_CC4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_EXT_IT15

ADC group regular conversion trigger from external peripheral: external interrupt line 15 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM1_CH1

ADC group regular conversion trigger from external peripheral: LPTIM1 channel 1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM2_CH1

ADC group regular conversion trigger from external peripheral: LPTIM2 channel 1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM3_CH1

ADC group regular conversion trigger from external peripheral: LPTIM3 channel 1 event. Trigger edge set to rising edge (default setting).

ADC_EXTERNALTRIG_LPTIM4_OUT

ADC group regular conversion trigger from external peripheral: LPTIM4 OUT event. Trigger edge set to rising edge (default setting).

ADC4_EXTERNALTRIG_T1_CC4

ADC group regular conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC4_EXTERNALTRIG_T1_TRGO2

ADC4_EXTERNALTRIG_T2_TRGO

ADC4_EXTERNALTRIG_T15_TRGO

ADC4_EXTERNALTRIG_T6_TRGO

ADC4_EXTERNALTRIG_LPTIM1_CH1

ADC4_EXTERNALTRIG_LPTIM3_CH2

ADC4_EXTERNALTRIG_EXT_IT15

ADC4 Regular Conversion Number Verification

IS_ADC4_REGULAR_NB_CONV

ADC sequencer scan mode

ADC_SCAN_DISABLE

Scan mode disabled

ADC_SCAN_ENABLE

Scan mode enabled

ADC_CHSELR_SQ2_TO_SQ8

ADCx SMPR1 fields

ADC_SMPR1_FIELDS

ADC States

HAL_ADC_STATE_RESET

Notes:

- ADC state machine is managed by bitfields, state must be compared with bit by bit. For example: " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_REG_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc1) & HAL_ADC_STATE_AWD1) != 0UL) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready for use

HAL_ADC_STATE_BUSY_INTERNAL

ADC is busy due to an internal process (initialization, calibration)

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_REG_BUSY

A conversion on ADC group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_REG_EOC

Conversion data available on group regular

HAL_ADC_STATE_REG_OVR

Overrun occurrence

HAL_ADC_STATE_REG_EOSMP

Not available on this STM32 series: End Of Sampling flag raised

HAL_ADC_STATE_INJ_BUSY

A conversion on ADC group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on (if feature available), multimode ADC master control (if feature available))

HAL_ADC_STATE_INJ_EOC

Conversion data available on group injected

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

ADC in multimode slave state, controlled by another ADC master (when feature available)

8 HAL ADC Extension Driver

8.1 ADCEx Firmware driver registers structures

8.1.1 ADC_InjOversamplingTypeDef

ADC_InjOversamplingTypeDef is defined in the stm32u5xx_hal_adc_ex.h

Data Fields

- ***uint32_t Ratio***
- ***uint32_t RightBitShift***

Field Documentation

- ***uint32_t ADC_InjOversamplingTypeDef::Ratio***
  Configures the oversampling ratio. In case of ADC1 or ADC2 (if available), this parameter can be in the range from 0 to 1023 In case of ADC4, this parameter can be a value of ***ADC_HAL_EC_OVS_RATIO***
- ***uint32_t ADC_InjOversamplingTypeDef::RightBitShift***
  Configures the division coefficient for the Oversampler. This parameter can be a value of ***ADC_HAL_EC_OVS_SHIFT***

8.1.2 ADC_InjectionConfTypeDef

ADC_InjectionConfTypeDef is defined in the stm32u5xx_hal_adc_ex.h

Data Fields

- ***uint32_t InjectedChannel***
- ***uint32_t InjectedRank***
- ***uint32_t InjectedSamplingTime***
- ***uint32_t InjectedSingleDiff***
- ***uint32_t InjectedOffsetNumber***
- ***uint32_t InjectedOffset***
- ***FunctionalState InjectedOffsetSignedSaturation***
- ***uint32_t InjectedOffsetSign***
- ***FunctionalState InjectedOffsetSaturation***
- ***uint32_t InjectedLeftBitShift***
- ***uint32_t InjectedNbrOfConversion***
- ***FunctionalState InjectedDiscontinuousConvMode***
- ***FunctionalState AutoInjectedConv***
- ***uint32_t ExternalTrigInjecConv***
- ***uint32_t ExternalTrigInjecConvEdge***
- ***FunctionalState InjecOversamplingMode***
- ***ADC_InjOversamplingTypeDef InjecOversampling***

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
  Specifies the channel to configure into ADC group injected. This parameter can be a value of ***ADC_HAL_EC_CHANNEL*** Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
  Specifies the rank in the ADC group injected sequencer. This parameter must be a value of ***ADC_INJ_SEQ_RANKS***. Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions adjusted)

- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits). This parameter can be a value of ***ADC_HAL_EC_CHANNEL_SAMPLINGTIME***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting); Refer to device datasheet for timings values.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSingleDiff***
Selection of single-ended or differential input. In differential mode: Differential measurement is between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically. This parameter must be a value of: ***ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING***. Caution: This parameter applies to a channel that can be used in a regular and/or injected group. It overwrites the last setting. Note: Refer to Reference Manual to ensure the selected channel is available in differential mode. Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case of another parameter update on the fly)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetNumber***
Selects the offset number. This parameter can be a value of ***ADC_HAL_EC_OFFSET_NB***. Caution: Only one offset is allowed per channel. This parameter overwrites the last setting.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
Defines the offset to be subtracted from the raw converted data. Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSignedSaturation***
Specify whether the Signed saturation feature is used or not. This parameter is only applied when InjectedOffsetSaturation is ENABLE. This parameter is applied only for 14-bit or 8-bit resolution. This parameter can be set to ENABLE or DISABLE. Note:
  – If InjectedOffsetSignedSaturation is set to DISABLE the unsigned saturation feature is used
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffsetSign***
Define if the offset should be subtracted (negative sign) or added (positive sign) from or to the raw converted data. This parameter can be a value of ***ADCEx_OffsetSign***. Note:
  – This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
  – This parameter is specific to ADC1 only.
- ***FunctionalState ADC_InjectionConfTypeDef::InjectedOffsetSaturation***
Define if the offset should be saturated upon under or over flow. This parameter value can be ENABLE or DISABLE. Note:
  – This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).
  – This parameter is specific to ADC1 only.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedLeftBitShift***
Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADCEx_Left_Bit_Shift***
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
Specifies the number of ranks that will be converted within the ADC group injected sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- ***FunctionalState ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode***
Specifies whether the conversions sequence of ADC group injected is performed in Complete-sequence/ Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). Note: For injected group, discontinuous mode converts the sequence channel by channel (discontinuous length fixed to 1 rank). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::AutoInjectedConv***
Enables or disables the selected ADC group injected automatic conversion after regular one This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE) Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_INJECTED_SOFTWARE_START) Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv***
Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled and software trigger is used instead. This parameter can be a value of ***ADC_injected_external_trigger_source***. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge***
Selects the external trigger edge of injected group. This parameter can be a value of ***ADC_injected_external_trigger_edge***. If trigger source is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEx_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- ***FunctionalState ADC_InjectionConfTypeDef::InjecOversamplingMode***
Specifies whether the oversampling feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).
- ***ADC_InjOversamplingTypeDef ADC_InjectionConfTypeDef::InjecOversampling***
Specifies the Oversampling parameters. Caution: this setting overwrites the previous oversampling configuration if oversampling already enabled. Note: This parameter can be modified only if there is no conversion is ongoing (both ADSTART and JADSTART cleared).

8.1.3 ADC_MultiModeTypeDef

ADC_MultiModeTypeDef is defined in the stm32u5xx_hal_adc_ex.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t DualModeData***
- ***uint32_t TwoSamplingDelay***

Field Documentation

- ***uint32_t ADC_MultiModeTypeDef::Mode***
Configures the ADC to operate in independent or multimode. This parameter can be a value of ***ADC_HAL_EC_MULTI_MODE***.
- ***uint32_t ADC_MultiModeTypeDef::DualModeData***
Configures the Dual ADC Mode Data Format: This parameter can be a value of ***ADCEx_Dual_Mode_Data_Format***

- ***uint32_t ADC_MultiModeTypeDef::TwoSamplingDelay***
  Configures the Delay between 2 sampling phases. This parameter can be a value of *ADC_HAL_EC_MULTI_TWOSMP_DELAY*. Delay range depends on selected resolution: from 1 to 12 clock cycles for 12 bits, from 1 to 10 clock cycles for 10 bits, from 1 to 8 clock cycles for 8 bits, from 1 to 6 clock cycles for 6 bits.

8.2 ADCEx Firmware driver API description

The following section lists the various functions of the ADCEx library.

8.2.1 IO operation functions

This section provides functions allowing to:
- Perform the ADC self-calibration for single or differential ending.
- Get calibration factors for single or differential ending.
- Set calibration factors for single or differential ending.
- Start conversion of ADC group injected.
- Stop conversion of ADC group injected.
- Poll for conversion complete on ADC group injected.
- Get result of ADC group injected channel conversion.
- Start conversion of ADC group injected and enable interruptions.
- Stop conversion of ADC group injected and disable interruptions.
- When multimode feature is available, start multimode and enable DMA transfer.
- Stop multimode and disable ADC DMA transfer.
- Get result of multimode conversion.

This section contains the following APIs:
- *HAL_ADCEx_Calibration_Start()*
- *HAL_ADCEx_Calibration_GetValue()*
- *HAL_ADCEx_LinearCalibration_GetValue()*
- *HAL_ADCEx_Calibration_SetValue()*
- *HAL_ADCEx_LinearCalibration_SetValue()*
- *HAL_ADCEx_InjectedStart()*
- *HAL_ADCEx_InjectedStop()*
- *HAL_ADCEx_InjectedPollForConversion()*
- *HAL_ADCEx_InjectedStart_IT()*
- *HAL_ADCEx_InjectedStop_IT()*
- *HAL_ADCEx_MultiModeStart_DMA()*
- *HAL_ADCEx_MultiModeStop_DMA()*
- *HAL_ADCEx_MultiModeGetValue()*
- *HAL_ADCEx_MultiModeStart_DMA_Data32()*
- *HAL_ADCEx_MultiModeGetValue_Data32()*
- *HAL_ADCEx_InjectedGetValue()*
- *HAL_ADCEx_InjectedConvCpltCallback()*
- *HAL_ADCEx_InjectedQueueOverflowCallback()*
- *HAL_ADCEx_LevelOutOfWindow2Callback()*
- *HAL_ADCEx_LevelOutOfWindow3Callback()*
- *HAL_ADCEx_EndOfSamplingCallback()*
- *HAL_ADCEx_RegularStop()*
- *HAL_ADCEx_RegularStop_IT()*
- *HAL_ADCEx_RegularStop_DMA()*
- *HAL_ADCEx_RegularMultiModeStop_DMA()*

8.2.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group
- Configure multimode when multimode feature is available
- Enable or Disable Injected Queue
- Disable ADC voltage regulator
- Enter ADC deep-power-down mode

This section contains the following APIs:

- *HAL_ADCEx_InjectedConfigChannel()*
- *HAL_ADCEx_MultiModeConfigChannel()*
- *HAL_ADCEx_DisableVoltageRegulator()*
- *HAL_ADCEx_EnterADCDeepPowerDownMode()*

8.2.3 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc, uint32_t CalibrationMode, uint32_t SingleDiff)

Function description

Perform an ADC automatic self-calibration Calibration prerequisite: ADC must be disabled (execute this function before HAL_ADC_Start() or after HAL_ADC_Stop()).

Parameters

- **hadc:** ADC handle
- **CalibrationMode:** Selection of calibration offset or linear calibration offset.
  – ADC_CALIB_OFFSET Channel in mode calibration offset
  – ADC_CALIB_OFFSET_LINEARITY Channel in mode linear calibration offset
- **SingleDiff:** Selection of single-ended or differential input This parameter can be one of the following values:
  – ADC_SINGLE_ENDED Channel in mode input single ended
  – ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **HAL:** status

HAL_ADCEx_Calibration_GetValue

Function name

uint32_t HAL_ADCEx_Calibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff)

Function description

Get the calibration factor.

Parameters

- **hadc:** ADC handle.
- **SingleDiff:** This parameter can be only:
  – ADC_SINGLE_ENDED Channel in mode input single ended
  – ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended

Return values

- **Calibration:** value.

HAL_ADCEx_LinearCalibration_GetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_GetValue (ADC_HandleTypeDef * hadc, uint32_t * pLinearCalib_Buffer)

Function description

Get the calibration factor from automatic conversion result.

Parameters

- **hadc:** ADC handle
- **pLinearCalib_Buffer:** Linear calibration factor (table of 9 elements)

Return values

- **HAL:** state

HAL_ADCEx_Calibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set the calibration factor to overwrite automatic conversion result.

Parameters

- **hadc:** ADC handle
- **SingleDiff:** This parameter can be only:
  – ADC_SINGLE_ENDED Channel in mode input single ended
  – ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
- **CalibrationFactor:** Calibration factor (range 0xFFFF for ADC1 and ADC2, range 0x7F for ADC4)

Return values

- **HAL:** state

HAL_ADCEx_LinearCalibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_LinearCalibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t * pLinearCalib_Buffer)

Function description

Set the linear calibration factor.

Parameters

- **hadc:** ADC handle
- **pLinearCalib_Buffer:** Linear calibration factor (table of 9 elements)

Return values

- **HAL:** state

HAL_ADCEx_InjectedStart

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- Interruptions enabled in this function: None.
- Case of multimode enabled when multimode feature is available: HAL_ADCEx_InjectedStart() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- In case of multimode enabled (when multimode feature is available), HAL_ADCEx_InjectedStop() must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).

HAL_ADCEx_InjectedPollForConversion

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for injected group conversion to be completed.

Parameters

- **hadc:** ADC handle
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- Depending on hadc->Init.EOCSelection, JEOS or JEOC is checked and cleared depending on AUTDLY bit status.

HAL_ADCEx_InjectedStart_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC, start conversion of injected group with interruption.

Parameters

- **hadc:** ADC handle.

Return values

- **HAL:** status.

Notes

- Interruptions enabled in this function according to initialization setting : JEOC (end of conversion) or JEOS (end of sequence)
- Case of multimode enabled (when multimode feature is enabled): HAL_ADCEx_InjectedStart_IT() API must be called for ADC slave first, then for ADC master. For ADC slave, ADC is enabled only (conversion is not started). For ADC master, ADC is enabled and multimode conversion is started.

HAL_ADCEx_InjectedStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop conversion of injected channels, disable interruption of end-of-conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- If injected group mode auto-injection is enabled, function HAL_ADC_Stop must be used.
- Case of multimode enabled (when multimode feature is available): HAL_ADCEx_InjectedStop_IT() API must be called for ADC master first, then for ADC slave. For ADC master, conversion is stopped and ADC is disabled. For ADC slave, ADC is disabled only (conversion stop of ADC master has already stopped conversion of ADC slave).
- In case of auto-injection mode, HAL_ADC_Stop() must be used.

HAL_ADCEx_MultiModeStart_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA (ADC_HandleTypeDef * hadc, const uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start MultiMode conversion and transfer regular results through DMA.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes).

Return values

- **HAL:** status

Notes

- Multimode must have been previously configured using HAL_ADCEx_MultiModeConfigChannel() function. Interruptions enabled in this function: overrun, DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- State field of Slave ADC handle is not updated in this configuration: user should not rely on it for information related to Slave regular conversions.

HAL_ADCEx_MultiModeStart_DMA_Data32

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA_Data32 (ADC_HandleTypeDef * hadc, const uint32_t * pData, uint32_t Length)

Function description

Enable ADC, start MultiMode conversion and transfer regular results through DMA.

Parameters

- **hadc:** ADC handle of ADC
- **pData:** Destination Buffer address.
- **Length:** Length of data to be transferred from ADC peripheral to memory (in bytes).

Return values

- **HAL:** status

Notes

- Multimode must have been previously configured using HAL_ADCEx_MultiModeConfigChannel() function. Interruptions enabled in this function: overrun, DMA half transfer, DMA transfer complete. Each of these interruptions has its dedicated callback function.
- State field of Slave ADC handle is not updated in this configuration: user should not rely on it for information related to Slave regular conversions.
- - Only ADC master could start the conversion. Two ADC conversions (Master & Slave) per external trig and so two DMA requests.

HAL_ADCEx_MultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. MultiMode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable Multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADC_Stop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_MultiModeGetValue

Function name

uint32_t HAL_ADCEx_MultiModeGetValue (const ADC_HandleTypeDef * hadc)

Function description

Return the last ADC Master and Slave regular conversions results when in multimode configuration.

Parameters

- **hadc:** ADC handle of ADC Master (handle of ADC Slave must not be used)

Return values

- **The:** converted data values.

HAL_ADCEx_MultiModeGetValue_Data32

Function name

uint32_t HAL_ADCEx_MultiModeGetValue_Data32 (const ADC_HandleTypeDef * hadc)

Function description

Return the last ADC Master and Slave regular conversions results when in multimode configuration.

Parameters

- **hadc:** ADC handle of ADC Master (handle of ADC Slave must not be used)

Return values

- **The:** converted data values.

HAL_ADCEx_InjectedGetValue

Function name

uint32_t HAL_ADCEx_InjectedGetValue (const ADC_HandleTypeDef * hadc, uint32_t InjectedRank)

Function description

Get ADC injected group conversion result.

Parameters

- **hadc:** ADC handle
- **InjectedRank:** the converted ADC injected rank. This parameter can be one of the following values:
  - ADC_INJECTED_RANK_1 ADC group injected rank 1
  - ADC_INJECTED_RANK_2 ADC group injected rank 2
  - ADC_INJECTED_RANK_3 ADC group injected rank 3
  - ADC_INJECTED_RANK_4 ADC group injected rank 4

Return values

- **ADC:** group injected conversion data

Notes

- Reading register JDRx automatically clears ADC flag JEOC (ADC group injected end of unitary conversion).
- This function does not clear ADC flag JEOS (ADC group injected end of sequence conversion) Occurrence of flag JEOS rising: If sequencer is composed of 1 rank, flag JEOS is equivalent to flag JEOC.If sequencer is composed of several ranks, during the scan sequence flag JEOC only is raised, at the end of the scan sequence both flags JEOC and EOS are raised. Flag JEOS must not be cleared by this function because it would not be compliant with low power features (feature low power auto-wait, not available on all STM32 families). To clear this flag, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADCEx_InjectedPollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_JEOS).

HAL_ADCEx_InjectedConvCpltCallback

Function name

void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)

Function description

Injected conversion complete callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_InjectedQueueOverflowCallback

Function name

void HAL_ADCEx_InjectedQueueOverflowCallback (ADC_HandleTypeDef * hadc)

Function description

Injected context queue overflow callback.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

Notes

- This callback is called if injected context queue is enabled (parameter "QueueInjectedContext" in injected channel configuration) and if a new injected context is set when queue is full (maximum 2 contexts).

HAL_ADCEx_LevelOutOfWindow2Callback

Function name

void HAL_ADCEx_LevelOutOfWindow2Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 2 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_LevelOutOfWindow3Callback

Function name

void HAL_ADCEx_LevelOutOfWindow3Callback (ADC_HandleTypeDef * hadc)

Function description

Analog watchdog 3 callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_EndOfSamplingCallback

Function name

void HAL_ADCEx_EndOfSamplingCallback (ADC_HandleTypeDef * hadc)

Function description

End Of Sampling callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

HAL_ADCEx_RegularStop

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_IT

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of ADC groups regular and injected, disable interrution of end-of-conversion, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADCEx_RegularStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral if no conversion is on going on injected group.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- HAL_ADCEx_RegularStop_DMA() function is dedicated to single-ADC mode only. For multimode (when multimode feature is available), HAL_ADCEx_RegularMultiModeStop_DMA() API must be used.

HAL_ADCEx_RegularMultiModeStop_DMA

Function name

HAL_StatusTypeDef HAL_ADCEx_RegularMultiModeStop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop DMA-based multimode ADC conversion, disable ADC DMA transfer, disable ADC peripheral if no injected conversion is on-going.

Parameters

- **hadc:** ADC handle of ADC master (handle of ADC slave must not be used)

Return values

- **HAL:** status

Notes

- Multimode is kept enabled after this function. Multimode DMA bits (MDMA and DMACFG bits of common CCR register) are maintained. To disable multimode (set with HAL_ADCEx_MultiModeConfigChannel()), ADC must be reinitialized using HAL_ADC_Init() or HAL_ADC_DeInit(), or the user can resort to HAL_ADCEx_DisableMultiMode() API.
- In case of DMA configured in circular mode, function HAL_ADCEx_RegularStop_DMA() must be called after this function with handle of ADC slave, to properly disable the DMA channel.

HAL_ADCEx_InjectedConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * pConfigInjected)

Function description

Configure a channel to be assigned to ADC group injected.

Parameters

- **hadc:** ADC handle
- **pConfigInjected:** Structure of ADC injected group and ADC channel for injected group.

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: Refer to comments of structure "ADC_InjectionConfTypeDef".
- In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit().
- Caution: For Injected Context Queue use, a context must be fully defined before start of injected conversion. All channels are configured consecutively for the same ADC instance. Therefore, the number of calls to HAL_ADCEx_InjectedConfigChannel() must be equal to the value of parameter InjectedNbrOfConversion for each context. Example 1: If 1 context is intended to be used (or if there is no use of the Injected Queue Context feature) and if the context contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel (i.e. 3 times) before starting a conversion. This function must not be called to configure a 4th injected channel: it would start a new context into context queue.Example 2: If 2 contexts are intended to be used and each of them contains 3 injected ranks (InjectedNbrOfConversion = 3), HAL_ADCEx_InjectedConfigChannel() must be called once for each channel and for each context (3 channels x 2 contexts = 6 calls). Conversion can start once the 1st context is set, that is after the first three HAL_ADCEx_InjectedConfigChannel() calls. The 2nd context can be set on the fly.

HAL_ADCEx_MultiModeConfigChannel

Function name

HAL_StatusTypeDef HAL_ADCEx_MultiModeConfigChannel (ADC_HandleTypeDef * hadc, ADC_MultiModeTypeDef * pMultimode)

Function description

Enable ADC multimode and configure multimode parameters.

Parameters

- **hadc:** Master ADC handle
- **pMultimode:** Structure of ADC multimode configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes multimode parameters, following calls to this function can be used to reconfigure some parameters of structure "ADC_MultiModeTypeDef" on the fly, without resetting the ADCs. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_MultiModeTypeDef".
- To move back configuration from multimode to single mode, ADC must be reset (using function HAL_ADC_Init()).

HAL_ADCEx_DisableVoltageRegulator

Function name

HAL_StatusTypeDef HAL_ADCEx_DisableVoltageRegulator (ADC_HandleTypeDef * hadc)

Function description

Disable ADC voltage regulator.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- Disabling voltage regulator allows to save power. This operation can be carried out only when ADC is disabled.
- To enable again the voltage regulator, the user is expected to resort to HAL_ADC_Init() API.

HAL_ADCEx_EnterADCDeepPowerDownMode

Function name

HAL_StatusTypeDef HAL_ADCEx_EnterADCDeepPowerDownMode (ADC_HandleTypeDef * hadc)

Function description

Enter ADC deep-power-down mode.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status

Notes

- This mode is achieved in setting DEEPPWD bit and allows to save power in reducing leakage currents. It is particularly interesting before entering stop modes.
- Setting DEEPPWD automatically clears ADVREGEN bit and disables the ADC voltage regulator. This means that this API encompasses HAL_ADCEx_DisableVoltageRegulator(). Additionally, the internal calibration is lost.
- To exit the ADC deep-power-down mode, the user is expected to resort to HAL_ADC_Init() API as well as to relaunch a calibration with HAL_ADCEx_Calibration_Start() API or to re-apply a previously saved calibration factor.

8.3 ADCEx Firmware driver defines

The following section lists the various define and macros of the module.

8.3.1 ADCEx

ADCEx

ADC indexes for linear calibration

ADC_CALIB_OFFSET_INDEX

Offset Calibration Index

ADC_CALIB_LINEARITY_INDEX1

Linearity Calibration Index 1

ADC_CALIB_LINEARITY_INDEX2

Linearity Calibration Index 1

ADC_CALIB_LINEARITY_INDEX3

Linearity Calibration Index 1

ADC_CALIB_LINEARITY_INDEX4

Linearity Calibration Index 1

ADC_CALIB_LINEARITY_INDEX5

Linearity Calibration Index 1

ADC_CALIB_LINEARITY_INDEX6

Linearity Calibration Index 1

ADC_CALIB_LINEARITY_INDEX7

Linearity Calibration Index 1

ADC_CALIB_INTEROFFSET_INDEX

Linearity Calibration Index 1

ADC Extended Calibration mode offset mode or linear mode

ADC_CALIB_OFFSET

ADC_CALIB_OFFSET_LINEARITY

ADC Extended Dual Mode Data Formatting

ADC_DUALMODEDATAFORMAT_DISABLED

Dual ADC mode without data packing: ADCx_CDR and ADCx_CDR2 registers not used

ADC_DUALMODEDATAFORMAT_32_10_BITS

Data formatting mode for 32 down to 10-bit resolution

ADC_DUALMODEDATAFORMAT_8_BITS

Data formatting mode for 8-bit resolution

ADC Extended Exported Macros

ADC_FORCE_MODE_INDEPENDENT

Description:

- Force ADC instance in multimode mode independent (multimode disable).

Parameters:

- __HANDLE__: ADC handle.

Return value:

- None

Notes:

- This macro must be used only in case of transition from multimode to mode independent and in case of unknown previous state, to ensure ADC configuration is in mode independent. Standard way of multimode configuration change is done from HAL ADC handle of ADC master using function "HAL_ADCEx_MultiModeConfigChannel(..., ADC_MODE_INDEPENDENT)")". Usage of this macro is not the Standard way of multimode configuration and can lead to have HAL ADC handles status misaligned. Usage of this macro must be limited to cases mentioned above.

ADC Extended Oversampling left Shift

ADC_LEFTBITSHIFT_NONE

ADC No bit shift

ADC_LEFTBITSHIFT_1

ADC 1 bit shift

ADC_LEFTBITSHIFT_2

ADC 2 bits shift

ADC_LEFTBITSHIFT_3

ADC 3 bits shift

ADC_LEFTBITSHIFT_4

ADC 4 bits shift

ADC_LEFTBITSHIFT_5

ADC 5 bits shift

ADC_LEFTBITSHIFT_6

ADC 6 bits shift

ADC_LEFTBITSHIFT_7

ADC 7 bits shift

ADC_LEFTBITSHIFT_8

ADC 8 bits shift

ADC_LEFTBITSHIFT_9

ADC 9 bits shift

ADC_LEFTBITSHIFT_10

ADC 10 bits shift

ADC_LEFTBITSHIFT_11

ADC 11 bits shift

ADC_LEFTBITSHIFT_12

ADC 12 bits shift

ADC_LEFTBITSHIFT_13

ADC 13 bits shift

ADC_LEFTBITSHIFT_14

ADC 14 bits shift

ADC_LEFTBITSHIFT_15

ADC 15 bits shift

ADC Extended Offset Sign

ADC_OFFSET_SIGN_NEGATIVE

Offset sign negative, offset is subtracted

ADC_OFFSET_SIGN_POSITIVE

Offset sign positive, offset is added

9 HAL COMP Generic Driver

9.1 COMP Firmware driver registers structures

9.1.1 COMP_InitTypeDef

COMP_InitTypeDef is defined in the stm32u5xx_hal_comp.h

Data Fields

- ***uint32_t WindowMode***
- ***uint32_t WindowOutput***
- ***uint32_t Mode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t Hysteresis***
- ***uint32_t OutputPol***
- ***uint32_t BlankingSrce***
- ***uint32_t TriggerMode***

Field Documentation

- ***uint32_t COMP_InitTypeDef::WindowMode***
  Set window mode of a pair of comparators instances (2 consecutive instances odd and even COMP<x> and COMP<x+1>). Note: HAL COMP driver allows to set window mode from any COMP instance of the pair of COMP instances composing window mode. This parameter can be a value of ***COMP_WindowMode***
- ***uint32_t COMP_InitTypeDef::WindowOutput***
  Set window mode output. This parameter can be a value of ***COMP_WindowOutput***
- ***uint32_t COMP_InitTypeDef::Mode***
  Set comparator operating mode to adjust power and speed. Note: For the characteristics of comparator power modes (propagation delay and power consumption), refer to device datasheet. This parameter can be a value of ***COMP_PowerMode***
- ***uint32_t COMP_InitTypeDef::InputPlus***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_InputPlus***
- ***uint32_t COMP_InitTypeDef::InputMinus***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_InputMinus***
- ***uint32_t COMP_InitTypeDef::Hysteresis***
  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_Hysteresis***
- ***uint32_t COMP_InitTypeDef::OutputPol***
  Set comparator output polarity. This parameter can be a value of ***COMP_OutputPolarity***
- ***uint32_t COMP_InitTypeDef::BlankingSrce***
  Set comparator blanking source. This parameter can be a value of ***COMP_BlankingSrce***
- ***uint32_t COMP_InitTypeDef::TriggerMode***
  Set the comparator output triggering External Interrupt Line (EXTI). This parameter can be a value of ***COMP_EXTI_TriggerMode***

9.1.2 __COMP_HandleTypeDef

__COMP_HandleTypeDef is defined in the stm32u5xx_hal_comp.h

Data Fields

- ***COMP_TypeDef * Instance***
- ***COMP_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_COMP_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TriggerCallback***

- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***COMP_TypeDef* __COMP_HandleTypeDef::Instance***
  Register base address
- ***COMP_InitTypeDef __COMP_HandleTypeDef::Init***
  COMP required parameters
- ***HAL_LockTypeDef __COMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_COMP_StateTypeDef __COMP_HandleTypeDef::State***
  COMP communication state
- ***__IO uint32_t __COMP_HandleTypeDef::ErrorCode***
  COMP error code
- ***void(* __COMP_HandleTypeDef::TriggerCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP trigger callback
- ***void(* __COMP_HandleTypeDef::MspInitCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP Msp Init callback
- ***void(* __COMP_HandleTypeDef::MspDeInitCallback)(struct __COMP_HandleTypeDef *hcomp)***
  COMP Msp DeInit callback

9.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

9.2.1 COMP Peripheral features

The STM32U5xx device family integrates two analog comparators instances: COMP1, COMP2 except for the products featuring only one instance: COMP1 (in this case, all comments related to pair of comparators are not applicable)

1. Comparators input minus (inverting input) and input plus (non inverting input) can be set to internal references or to GPIO pins (refer to GPIO list in reference manual).
2. Comparators output level is available using HAL_COMP_GetOutputLevel() and can be redirected to other peripherals: GPIO pins (in mode alternate functions for comparator), timers. (refer to GPIO list in reference manual).
3. The comparators have interrupt capability through the EXTI controller with wake-up from sleep and stop modes.
   – COMP1 is internally connected to EXTI Line 20
   – COMP2 is internally connected to EXTI Line 21
4. Pairs of comparators instances can be combined in window mode (2 consecutive instances odd and even COMP<x> and COMP<x+1>). From the corresponding IRQ handler, the right interrupt source can be retrieved using macro __HAL_COMP_COMPx_EXTI_GET_FLAG().

9.2.2 How to use this driver

This driver provides functions to configure and program the comparator instances of STM32U5xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the HAL_COMP_MspInit():
   – Configure the GPIO connected to comparator inputs plus and minus in analog mode using HAL_GPIO_Init().
   – If needed, configure the GPIO connected to comparator output in alternate function mode using HAL_GPIO_Init().
   – If required enable the COMP interrupt by configuring and enabling EXTI line in Interrupt mode and selecting the desired sensitivity level using HAL_GPIO_Init() function. After that enable the comparator interrupt vector using HAL_NVIC_EnableIRQ() function.

2. Configure the comparator using HAL_COMP_Init() function:
   – Select the input minus (inverting input)
   – Select the input plus (non-inverting input)
   – Select the hysteresis
   – Select the blanking source
   – Select the output polarity
   – Select the power mode
   – Select the window mode

Note: HAL_COMP_Init() calls internally __HAL_RCC_SYSCFG_CLK_ENABLE() to enable internal control clock of the comparators. However, this is a legacy strategy. In future STM32 families, COMP clock enable must be implemented by user in "HAL_COMP_MspInit()". Therefore, for compatibility anticipation, it is recommended to implement __HAL_RCC_SYSCFG_CLK_ENABLE() in "HAL_COMP_MspInit()".

3. Reconfiguration on-the-fly of comparator can be done by calling again function HAL_COMP_Init() with new input structure parameters values.
4. Enable the comparator using HAL_COMP_Start() function.
5. Use HAL_COMP_TriggerCallback() or HAL_COMP_GetOutputLevel() functions to manage comparator outputs (events and output level).
6. Disable the comparator using HAL_COMP_Stop() function.
7. De-initialize the comparator using HAL_COMP_DeInit() function.
8. For safety purpose, comparator configuration can be locked using HAL_COMP_Lock() function. The only way to unlock the comparator is a device hardware reset.

Callback registration

The compilation flag USE_HAL_COMP_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_COMP_RegisterCallback() to register an interrupt callback.

Function HAL_COMP_RegisterCallback() allows to register following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_COMP_UnRegisterCallback to reset a callback to the default weak function.

HAL_COMP_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TriggerCallback : callback for COMP trigger.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_COMP_Init() and when the state is HAL_COMP_STATE_RESET all callbacks are set to the corresponding weak functions: example HAL_COMP_TriggerCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_COMP_Init()/ HAL_COMP_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_COMP_Init()/ HAL_COMP_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_COMP_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_COMP_STATE_READY or HAL_COMP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_COMP_RegisterCallback() before calling HAL_COMP_DeInit() or HAL_COMP_Init() function.

When the compilation flag USE_HAL_COMP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

9.2.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:
- *HAL_COMP_Init()*
- *HAL_COMP_DeInit()*
- *HAL_COMP_MspInit()*
- *HAL_COMP_MspDeInit()*
- *HAL_COMP_RegisterCallback()*
- *HAL_COMP_UnRegisterCallback()*

9.2.4 IO operation functions

This section provides functions allowing to:
- Start a comparator instance.
- Stop a comparator instance.

This section contains the following APIs:
- *HAL_COMP_Start()*
- *HAL_COMP_Stop()*
- *HAL_COMP_IRQHandler()*

9.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the comparators.

This section contains the following APIs:
- *HAL_COMP_Lock()*
- *HAL_COMP_GetOutputLevel()*
- *HAL_COMP_TriggerCallback()*

9.2.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:
- *HAL_COMP_GetState()*
- *HAL_COMP_GetError()*

9.2.7 Detailed description of functions

HAL_COMP_Init

Function name

HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP according to the specified parameters in the COMP_InitTypeDef and initialize the associated handle.

Parameters
- **hcomp:** COMP handle

Return values
- **HAL:** status

Notes
- If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_COMP_DeInit

Function name

HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP peripheral.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- Deinitialization cannot be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

HAL_COMP_MspInit

Function name

void HAL_COMP_MspInit (COMP_HandleTypeDef * hcomp)

Function description

Initialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_MspDeInit

Function name

void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)

Function description

DeInitialize the COMP MSP.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_COMP_RegisterCallback (COMP_HandleTypeDef * hcomp, HAL_COMP_CallbackIDTypeDef CallbackID, pCOMP_CallbackTypeDef pCallback)

Function description

Register a User COMP Callback To be used instead of the weak predefined callback.

Parameters

- **hcomp:** Pointer to a COMP_HandleTypeDef structure that contains the configuration information for the specified COMP.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_COMP_TRIGGER_CB_ID Trigger callback ID
  – HAL_COMP_MSPINIT_CB_ID MspInit callback ID
  – HAL_COMP_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_COMP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_COMP_UnRegisterCallback (COMP_HandleTypeDef * hcomp, HAL_COMP_CallbackIDTypeDef CallbackID)

Function description

Unregister a COMP Callback COMP callback is redirected to the weak predefined callback.

Parameters

- **hcomp:** Pointer to a COMP_HandleTypeDef structure that contains the configuration information for the specified COMP.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_COMP_TRIGGER_CB_ID Trigger callback ID
  – HAL_COMP_MSPINIT_CB_ID MspInit callback ID
  – HAL_COMP_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_COMP_Start

Function name

HAL_StatusTypeDef HAL_COMP_Start (COMP_HandleTypeDef * hcomp)

Function description

Start the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_Stop

Function name

HAL_StatusTypeDef HAL_COMP_Stop (COMP_HandleTypeDef * hcomp)

Function description

Stop the comparator.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

HAL_COMP_IRQHandler

Function name

void HAL_COMP_IRQHandler (COMP_HandleTypeDef * hcomp)

Function description

Comparator IRQ handler.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_Lock

Function name

HAL_StatusTypeDef HAL_COMP_Lock (COMP_HandleTypeDef * hcomp)

Function description

Lock the selected comparator configuration.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** status

Notes

- A system reset is required to unlock the comparator configuration.
- Locking the comparator from reset state is possible if __HAL_RCC_SYSCFG_CLK_ENABLE() is being called before.

HAL_COMP_GetOutputLevel

Function name

uint32_t HAL_COMP_GetOutputLevel (const COMP_HandleTypeDef * hcomp)

Function description

Return the output level (high or low) of the selected comparator.

HAL_COMP_TriggerCallback

Function name

void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)

Function description

Comparator trigger callback.

Parameters

- **hcomp:** COMP handle

Return values

- **None:**

HAL_COMP_GetState

Function name

HAL_COMP_StateTypeDef HAL_COMP_GetState (const COMP_HandleTypeDef * hcomp)

Function description

Return the COMP handle state.

Parameters

- **hcomp:** COMP handle

Return values

- **HAL:** state

HAL_COMP_GetError

Function name

uint32_t HAL_COMP_GetError (const COMP_HandleTypeDef * hcomp)

Function description

Return the COMP error code.

Parameters

- **hcomp:** COMP handle

Return values

- **COMP:** error code

9.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

9.3.1 COMP

COMP

COMP blanking source

COMP_BLANKINGSRC_NONE

Comparator output without blanking

COMP_BLANKINGSRC_TIM1_OC5

TIM1 OC5 selected as blanking source for COMP1

COMP_BLANKINGSRC_TIM2_OC3

TIM2 OC3 selected as blanking source for COMP1

COMP_BLANKINGSRC_TIM3_OC3

TIM3 OC3 selected as blanking source for COMP1

COMP_BLANKINGSRC_TIM3_OC4

TIM3 OC4 selected as blanking source for COMP2

COMP_BLANKINGSRC_TIM8_OC5

TIM8 OC5 selected as blanking source for COMP2

COMP_BLANKINGSRC_TIM15_OC1

TIM15 OC1 selected as blanking source for COMP2

COMP Error Code

HAL_COMP_ERROR_NONE

No error

HAL_COMP_ERROR_INVALID_CALLBACK

Invalid Callback error

COMP Exported Types

COMP_STATE_BITFIELD_LOCK

COMP EXTI Lines

COMP_EXTI_LINE_COMP1

EXTI line 17 connected to COMP1 output

COMP_EXTI_LINE_COMP2

EXTI line 18 connected to COMP2 output

COMP_EXTI_IT

EXTI line event with interruption

COMP_EXTI_EVENT

EXTI line event only (without interruption)

COMP_EXTI_RISING

EXTI line event on rising edge

COMP_EXTI_FALLING

EXTI line event on falling edge

COMP external interrupt line management

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_IT

Description:

- Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_IT

Description:

- Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP1 EXTI line.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_EVENT

Description:

- Enable the COMP1 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_EVENT

Description:

- Disable the COMP1 EXTI line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GET_RISING_FLAG

Description:
- Check whether the COMP1 EXTI line rising flag is set or not.

Return value:
- RESET: or SET

__HAL_COMP_COMP1_EXTI_GET_FALLING_FLAG

Description:
- Check whether the COMP1 EXTI line falling flag is set or not.

Return value:
- RESET: or SET

__HAL_COMP_COMP1_EXTI_CLEAR_RISING_FLAG

Description:
- Clear the COMP1 EXTI raising flag.

Return value:
- None

__HAL_COMP_COMP1_EXTI_CLEAR_FALLING_FLAG

Description:
- Clear the COMP1 EXTI falling flag.

Return value:
- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE

Description:
- Enable the COMP2 EXTI line rising edge trigger.

Return value:
- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE

Description:
- Disable the COMP2 EXTI line rising edge trigger.

Return value:
- None

__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE

Description:
- Enable the COMP2 EXTI line falling edge trigger.

Return value:
- None

__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE

Description:
- Disable the COMP2 EXTI line falling edge trigger.

Return value:
- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:
- Enable the COMP2 EXTI line rising & falling edge trigger.

Return value:
- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:
- Disable the COMP2 EXTI line rising & falling edge trigger.

Return value:
- None

__HAL_COMP_COMP2_EXTI_ENABLE_IT

Description:
- Enable the COMP2 EXTI line in interrupt mode.

Return value:
- None

__HAL_COMP_COMP2_EXTI_DISABLE_IT

Description:
- Disable the COMP2 EXTI line in interrupt mode.

Return value:
- None

__HAL_COMP_COMP2_EXTI_ENABLE_EVENT

Description:
- Enable the COMP2 EXTI Line in event mode.

Return value:
- None

__HAL_COMP_COMP2_EXTI_DISABLE_EVENT

Description:
- Disable the COMP2 EXTI Line in event mode.

Return value:
- None

__HAL_COMP_COMP2_EXTI_GET_RISING_FLAG

Description:
- Check whether the COMP2 EXTI line raising flag is set or not.

Return value:
- RESET: or SET

__HAL_COMP_COMP2_EXTI_GET_FALLING_FLAG

Description:
- Check whether the COMP2 EXTI line falling flag is set or not.

Return value:
- RESET: or SET

__HAL_COMP_COMP2_EXTI_CLEAR_RISING_FLAG

Description:

- Clear the the COMP2 EXTI raising flag.

Return value:

- None

__HAL_COMP_COMP2_EXTI_CLEAR_FALLING_FLAG

Description:

- Clear the the COMP2 EXTI falling flag.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GENERATE_SWIT

Description:

- Generate a software interrupt on the COMP2 EXTI line.

Return value:

- None

COMP output to EXTI

COMP_TRIGGERMODE_NONE

Comparator output triggering no External Interrupt Line

COMP_TRIGGERMODE_IT_RISING

Comparator output triggering External Interrupt Line event with interruption, on rising edge

COMP_TRIGGERMODE_IT_FALLING

Comparator output triggering External Interrupt Line event with interruption, on falling edge

COMP_TRIGGERMODE_IT_RISING_FALLING

Comparator output triggering External Interrupt Line event with interruption, on both rising and falling edges

COMP_TRIGGERMODE_EVENT_RISING

Comparator output triggering External Interrupt Line event only (without interruption), on rising edge

COMP_TRIGGERMODE_EVENT_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on falling edge

COMP_TRIGGERMODE_EVENT_RISING_FALLING

Comparator output triggering External Interrupt Line event only (without interruption), on both rising and falling edges

COMP private macros to get EXTI line associated with comparators

COMP_GET_EXTI_LINE

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- __INSTANCE__: specifies the COMP instance.

Return value:

- value: of

COMP Handle Management

__HAL_COMP_RESET_HANDLE_STATE

Description:
- Reset COMP handle state.

Parameters:
- __HANDLE__: COMP handle

Return value:
- None

COMP_CLEAR_ERRORCODE

Description:
- Clear COMP error code (set it to no error code "HAL_COMP_ERROR_NONE").

Parameters:
- __HANDLE__: COMP handle

Return value:
- None

__HAL_COMP_ENABLE

Description:
- Enable the specified comparator.

Parameters:
- __HANDLE__: COMP handle

Return value:
- None

__HAL_COMP_DISABLE

Description:
- Disable the specified comparator.

Parameters:
- __HANDLE__: COMP handle

Return value:
- None

__HAL_COMP_LOCK

Description:
- Lock the specified comparator configuration.

Parameters:
- __HANDLE__: COMP handle

Return value:
- None

Notes:
- Using this macro induce HAL COMP handle state machine being no more in line with COMP instance state. To keep HAL COMP handle state machine updated, it is recommended to use function "HAL_COMP_Lock')".

__HAL_COMP_IS_LOCKED

Description:
- Check whether the specified comparator is locked.

Parameters:
- __HANDLE__: COMP handle

Return value:
- Value: 0 if COMP instance is not locked, value 1 if COMP instance is locked

COMP hysteresis

COMP_HYSTERESIS_NONE

No hysteresis

COMP_HYSTERESIS_LOW

Hysteresis level low

COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

COMP_HYSTERESIS_HIGH

Hysteresis level high

COMP input minus (inverting input)

COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)

COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PB1 for COMP1, pin PB7 for COMP2)

COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PC4 for COMP1, pin PB3 for COMP2)

COMP input plus (non-inverting input)

COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PC5 for COMP1, pin PB4 for COMP2)

COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB2 for COMP1, pin PB6 for COMP2)

COMP_INPUT_PLUS_IO3

Comparator input plus connected to IO3 (pin PA2 for COMP1)

COMP Output Level

COMP_OUTPUT_LEVEL_LOW

COMP_OUTPUT_LEVEL_HIGH

COMP output Polarity

COMP_OUTPUTPOL_NONINVERTED

COMP output level is not inverted (comparator output is high when the input plus is at a higher voltage than the input minus)

COMP_OUTPUTPOL_INVERTED

COMP output level is inverted (comparator output is low when the input plus is at a higher voltage than the input minus)

COMP power mode

COMP_POWERMODE_HIGHSPEED

High Speed

COMP_POWERMODE_MEDIUMSPEED

Medium Speed

COMP_POWERMODE_ULTRALOWPOWER

Ultra-low power

COMP Window Mode

COMP_WINDOWMODE_DISABLE

Window mode disable: Comparators instances pair COMP1 and COMP2 are independent

COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

COMP_WINDOWMODE_COMP2_INPUT_PLUS_COMMON

Window mode enable: if used from COMP1 or COMP2 instance, comparators instances pair COMP1 and COMP2 have their input plus connected together, the common input is COMP2 input plus (COMP1 input plus is no more accessible).

COMP window mode instance differentiator

COMP_WINDOWMODE_COMP2

COMP window mode using common input of COMP instance: COMP2

COMP Window output

COMP_WINDOWOUTPUT_EACH_COMP

Window output default mode: Comparators output are indicating each their own state. To know window mode state: each comparator output must be read, if "((COMPx exclusive or COMPy) == 1)" then monitored signal is within comparators window.

COMP_WINDOWOUTPUT_COMP1

Window output synthesized on COMP1 output: COMP1 output is no more indicating its own state, but global window mode state (logical high means monitored signal is within comparators window). Note: impacts only comparator output signal level (COMPx_OUT propagated to GPIO, EXTI lines, timers, ...), does not impact output digital state of comparator (COMPx_VALUE) always reflecting each comparator output state.

COMP_WINDOWOUTPUT_COMP2

Window output synthesized on COMP2 output: COMP2 output is no more indicating its own state, but global window mode state (logical high means monitored signal is within comparators window). Note: impacts only comparator output signal level (COMPx_OUT propagated to GPIO, EXTI lines, timers, ...), does not impact output digital state of comparator (COMPx_VALUE) always reflecting each comparator output state.

COMP_WINDOWOUTPUT_BOTH

Window output synthesized on both comparators output of pair of comparator selected (COMP1 and COMP2: both comparators outputs are no more indicating their own state, but global window mode state (logical high means monitored signal is within comparators window). This is a specific configuration (technically possible but not relevant from application point of view: 2 comparators output used for the same signal level), standard configuration for window mode is one of the settings above.

10 HAL CORDIC Generic Driver

10.1 CORDIC Firmware driver registers structures

10.1.1 __CORDIC_HandleTypeDef

__CORDIC_HandleTypeDef is defined in the stm32u5xx_hal_cordic.h

Data Fields

- ***CORDIC_TypeDef * Instance***
- ***const int32_t * pInBuff***
- ***int32_t * pOutBuff***
- ***uint32_t NbCalcToOrder***
- ***uint32_t NbCalcToGet***
- ***uint32_t DMADirection***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CORDIC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* CalculateCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***CORDIC_TypeDef* __CORDIC_HandleTypeDef::Instance***
  Register base address
- ***const int32_t* __CORDIC_HandleTypeDef::pInBuff***
  Pointer to CORDIC input data buffer
- ***int32_t* __CORDIC_HandleTypeDef::pOutBuff***
  Pointer to CORDIC output data buffer
- ***uint32_t __CORDIC_HandleTypeDef::NbCalcToOrder***
  Remaining number of calculation to order
- ***uint32_t __CORDIC_HandleTypeDef::NbCalcToGet***
  Remaining number of calculation result to get
- ***uint32_t __CORDIC_HandleTypeDef::DMADirection***
  Direction of CORDIC DMA transfers
- ***DMA_HandleTypeDef* __CORDIC_HandleTypeDef::hdmaIn***
  CORDIC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* __CORDIC_HandleTypeDef::hdmaOut***
  CORDIC peripheral output data DMA handle parameters
- ***HAL_LockTypeDef __CORDIC_HandleTypeDef::Lock***
  CORDIC locking object
- ***__IO HAL_CORDIC_StateTypeDef __CORDIC_HandleTypeDef::State***
  CORDIC state
- ***__IO uint32_t __CORDIC_HandleTypeDef::ErrorCode***
  CORDIC peripheral error code This parameter can be a value of ***CORDIC_Error_Code***
- ***void(* __CORDIC_HandleTypeDef::ErrorCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC error callback
- ***void(* __CORDIC_HandleTypeDef::CalculateCpltCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC calculate complete callback

- ***void(* __CORDIC_HandleTypeDef::MspInitCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC Msp Init callback
- ***void(* __CORDIC_HandleTypeDef::MspDeInitCallback)(struct __CORDIC_HandleTypeDef *hcordic)***
  CORDIC Msp DeInit callback

10.1.2 CORDIC_ConfigTypeDef

CORDIC_ConfigTypeDef is defined in the stm32u5xx_hal_cordic.h

Data Fields

- ***uint32_t Function***
- ***uint32_t Scale***
- ***uint32_t InSize***
- ***uint32_t OutSize***
- ***uint32_t NbWrite***
- ***uint32_t NbRead***
- ***uint32_t Precision***

Field Documentation

- ***uint32_t CORDIC_ConfigTypeDef::Function***
  Function This parameter can be a value of ***CORDIC_Function***
- ***uint32_t CORDIC_ConfigTypeDef::Scale***
  Scaling factor This parameter can be a value of ***CORDIC_Scale***
- ***uint32_t CORDIC_ConfigTypeDef::InSize***
  Width of input data This parameter can be a value of ***CORDIC_In_Size***
- ***uint32_t CORDIC_ConfigTypeDef::OutSize***
  Width of output data This parameter can be a value of ***CORDIC_Out_Size***
- ***uint32_t CORDIC_ConfigTypeDef::NbWrite***
  Number of 32-bit write expected for one calculation This parameter can be a value of ***CORDIC_Nb_Write***
- ***uint32_t CORDIC_ConfigTypeDef::NbRead***
  Number of 32-bit read expected after one calculation This parameter can be a value of ***CORDIC_Nb_Read***
- ***uint32_t CORDIC_ConfigTypeDef::Precision***
  Number of cycles for calculation This parameter can be a value of ***CORDIC_Precision_In_Cycles_Number***

10.2 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

10.2.1 How to use this driver

The CORDIC HAL driver can be used as follows:

1. Initialize the CORDIC low level resources by implementing the HAL_CORDIC_MspInit():
   - Enable the CORDIC interface clock using __HAL_RCC_CORDIC_CLK_ENABLE()
   - In case of using interrupts (e.g. HAL_CORDIC_Calculate_IT())
     - Configure the CORDIC interrupt priority using HAL_NVIC_SetPriority()
     - Enable the CORDIC IRQ handler using HAL_NVIC_EnableIRQ()
     - In CORDIC IRQ handler, call HAL_CORDIC_IRQHandler()
   - In case of using DMA to control data transfer (e.g. HAL_CORDIC_Calculate_DMA())
     - Enable the DMA2 interface clock using __HAL_RCC_DMA2_CLK_ENABLE()
     - Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
     - Associate the initialized DMA handle to the CORDIC DMA handle using __HAL_LINKDMA()
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. Resort to HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()

2. Initialize the CORDIC HAL using HAL_CORDIC_Init(). This function
   – resorts to HAL_CORDIC_MspInit() for low-level initialization,
3. Configure CORDIC processing (calculation) using HAL_CORDIC_Configure(). This function configures:
   – Processing functions: Cosine, Sine, Phase, Modulus, Arctangent, Hyperbolic cosine, Hyperbolic sine, Hyperbolic arctangent, Natural log, Square root
   – Scaling factor: 1 to 2exp(-7)
   – Width of input data: 32 bits input data size (Q1.31 format) or 16 bits input data size (Q1.15 format)
   – Width of output data: 32 bits output data size (Q1.31 format) or 16 bits output data size (Q1.15 format)
   – Number of 32-bit write expected for one calculation: One 32-bits write or Two 32-bit write
   – Number of 32-bit read expected after one calculation: One 32-bits read or Two 32-bit read
   – Precision: 1 to 15 cycles for calculation (the more cycles, the better precision)
4. Four processing (calculation) functions are available:
   – Polling mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished API is HAL_CORDIC_Calculate
   – Polling Zero-overhead mode: processing API is blocking function i.e. it processes the data and wait till the processing is finished A bit faster than standard polling mode, but blocking also AHB bus API is HAL_CORDIC_CalculateZO
   – Interrupt mode: processing API is not blocking functions i.e. it processes the data under interrupt API is HAL_CORDIC_Calculate_IT
   – DMA mode: processing API is not blocking functions and the CPU is not used for data transfer, i.e. the data transfer is ensured by DMA API is HAL_CORDIC_Calculate_DMA
5. Call HAL_CORDIC_DeInit() to de-initialize the CORDIC peripheral. This function
   – resorts to HAL_CORDIC_MspDeInit() for low-level de-initialization,

Callback registration

10.2.2 Initialization and de-initialization functions

This section provides functions allowing to:
- Initialize the CORDIC peripheral and the associated handle
- DeInitialize the CORDIC peripheral
- Initialize the CORDIC MSP (MCU Specific Package)
- De-Initialize the CORDIC MSP

This section contains the following APIs:
- *HAL_CORDIC_Init()*
- *HAL_CORDIC_DeInit()*
- *HAL_CORDIC_MspInit()*
- *HAL_CORDIC_MspDeInit()*
- *HAL_CORDIC_RegisterCallback()*
- *HAL_CORDIC_UnRegisterCallback()*
- *HAL_CORDIC_RegisterCallback()*

10.2.3 Peripheral Control functions

This section provides functions allowing to:
- Configure the CORDIC peripheral: function, precision, scaling factor, number of input data and output data, size of input data and output data.
- Calculate output data of CORDIC processing on input date, using the existing CORDIC configuration

Four processing functions are available for calculation:
- Polling mode
- Polling mode, with Zero-Overhead register access
- Interrupt mode
- DMA mode

This section contains the following APIs:
- *HAL_CORDIC_Configure()*
- *HAL_CORDIC_Calculate()*
- *HAL_CORDIC_CalculateZO()*
- *HAL_CORDIC_Calculate_IT()*
- *HAL_CORDIC_Calculate_DMA()*

10.2.4 Callback functions

This section provides Interruption and DMA callback functions:
- DMA or Interrupt calculate complete
- DMA or Interrupt error

This section contains the following APIs:
- *HAL_CORDIC_ErrorCallback()*
- *HAL_CORDIC_CalculateCpltCallback()*

10.2.5 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:
- *HAL_CORDIC_IRQHandler()*

10.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:
- *HAL_CORDIC_GetState()*
- *HAL_CORDIC_GetError()*

10.2.7 Detailed description of functions

HAL_CORDIC_Init

Function name

HAL_StatusTypeDef HAL_CORDIC_Init (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC peripheral and the associated handle.

Parameters
- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values
- **HAL:** status

HAL_CORDIC_DeInit

Function name

HAL_StatusTypeDef HAL_CORDIC_DeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC peripheral.

Parameters
- **hcordic:** pointer to a CORDIC_HandleTypeDef structure.

Return values

- **HAL:** status

HAL_CORDIC_MspInit

Function name

void HAL_CORDIC_MspInit (CORDIC_HandleTypeDef * hcordic)

Function description

Initialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_MspDeInit

Function name

void HAL_CORDIC_MspDeInit (CORDIC_HandleTypeDef * hcordic)

Function description

DeInitialize the CORDIC MSP.

Parameters

- **hcordic:** CORDIC handle

Return values

- **None:**

HAL_CORDIC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CORDIC_RegisterCallback (CORDIC_HandleTypeDef * hcordic, HAL_CORDIC_CallbackIDTypeDef CallbackID, pCORDIC_CallbackTypeDef pCallback)

Function description

HAL_CORDIC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CORDIC_UnRegisterCallback (CORDIC_HandleTypeDef * hcordic, HAL_CORDIC_CallbackIDTypeDef CallbackID)

Function description

Unregister a CORDIC CallBack.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_CORDIC_ERROR_CB_ID error Callback ID
  – HAL_CORDIC_CALCULATE_CPLT_CB_ID calculate complete Callback ID
  – HAL_CORDIC_MSPINIT_CB_ID MspInit callback ID
  – HAL_CORDIC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_CORDIC_Configure

Function name

HAL_StatusTypeDef HAL_CORDIC_Configure (CORDIC_HandleTypeDef * hcordic, const CORDIC_ConfigTypeDef * sConfig)

Function description

Configure the CORDIC processing according to the specified parameters in the CORDIC_ConfigTypeDef structure.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module
- **sConfig:** pointer to a CORDIC_ConfigTypeDef structure that contains the CORDIC configuration information.

Return values

- **HAL:** status

HAL_CORDIC_Calculate

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in polling mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_CalculateZO

Function name

HAL_StatusTypeDef HAL_CORDIC_CalculateZO (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t Timeout)

Function description

Carry out data of CORDIC processing in Zero-Overhead mode (output data being read soon as input data are written), according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CORDIC_Calculate_IT

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_IT (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc)

Function description

Carry out data of CORDIC processing in interrupt mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.

Return values

- **HAL:** status

HAL_CORDIC_Calculate_DMA

Function name

HAL_StatusTypeDef HAL_CORDIC_Calculate_DMA (CORDIC_HandleTypeDef * hcordic, const int32_t * pInBuff, int32_t * pOutBuff, uint32_t NbCalc, uint32_t DMADirection)

Function description

Carry out input and/or output data of CORDIC processing in DMA mode, according to the existing CORDIC configuration.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module.
- **pInBuff:** Pointer to buffer containing input data for CORDIC processing.
- **pOutBuff:** Pointer to buffer where output data of CORDIC processing will be stored.
- **NbCalc:** Number of CORDIC calculation to process.
- **DMADirection:** Direction of DMA transfers. This parameter can be one of the following values:
  - CORDIC DMA direction CORDIC DMA direction

Return values

- **HAL:** status

Notes

- pInBuff or pOutBuff is unused in case of unique DMADirection transfer, and can be set to NULL value in this case.
- pInBuff and pOutBuff buffers must be 32-bit aligned to ensure a correct DMA transfer to and from the Peripheral.

HAL_CORDIC_ErrorCallback

Function name

void HAL_CORDIC_ErrorCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC error callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_CalculateCpltCallback

Function name

void HAL_CORDIC_CalculateCpltCallback (CORDIC_HandleTypeDef * hcordic)

Function description

CORDIC calculate complete callback.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_IRQHandler

Function name

void HAL_CORDIC_IRQHandler (CORDIC_HandleTypeDef * hcordic)

Function description

Handle CORDIC interrupt request.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **None:**

HAL_CORDIC_GetState

Function name

HAL_CORDIC_StateTypeDef HAL_CORDIC_GetState (const CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC handle state.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **HAL:** state

HAL_CORDIC_GetError

Function name

uint32_t HAL_CORDIC_GetError (const CORDIC_HandleTypeDef * hcordic)

Function description

Return the CORDIC peripheral error.

Parameters

- **hcordic:** pointer to a CORDIC_HandleTypeDef structure that contains the configuration information for CORDIC module

Return values

- **Error:** bit-map

Notes

- The returned error is a bit-map combination of possible errors

10.3 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

10.3.1 CORDIC

CORDIC

DMA Read Request Enable bit

CORDIC_DMA_REN

DMA Read requests enable

DMA Write Request Enable bit

CORDIC_DMA_WEN

DMA Write channel enable

CORDIC DMA direction

CORDIC_DMA_DIR_NONE

DMA direction : none

CORDIC_DMA_DIR_IN

DMA direction : Input of CORDIC

CORDIC_DMA_DIR_OUT

DMA direction : Output of CORDIC

CORDIC_DMA_DIR_IN_OUT

DMA direction : Input and Output of CORDIC

CORDIC Error code

HAL_CORDIC_ERROR_NONE

No error

HAL_CORDIC_ERROR_PARAM

Wrong parameter error

HAL_CORDIC_ERROR_NOT_READY

Peripheral not ready

HAL_CORDIC_ERROR_TIMEOUT

Timeout error

HAL_CORDIC_ERROR_DMA

DMA error

HAL_CORDIC_ERROR_INVALID_CALLBACK

Invalid Callback error

CORDIC Exported Macros

__HAL_CORDIC_RESET_HANDLE_STATE

Description:

- Reset CORDIC handle state.

Parameters:

- __HANDLE__: CORDIC handle

Return value:

- None

__HAL_CORDIC_ENABLE_IT

Description:

- Enable the CORDIC interrupt when result is ready.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_DISABLE_IT

Description:

- Disable the CORDIC interrupt.

Parameters:

- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC Interrupt. This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:

- None

__HAL_CORDIC_GET_IT

Description:
- Check whether the specified CORDIC interrupt occurred or not.

Parameters:
- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check

Return value:
- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_CORDIC_CLEAR_IT

Description:
- Clear specified CORDIC interrupt status.

Parameters:
- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to clear

Return value:
- None

__HAL_CORDIC_GET_FLAG

Description:
- Check whether the specified CORDIC status flag is set or not.

Parameters:
- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to check This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:
- SET: (flag is set) or RESET (flag is reset)

__HAL_CORDIC_CLEAR_FLAG

Description:
- Clear specified CORDIC status flag.

Parameters:
- __HANDLE__: CORDIC handle.
- __FLAG__: CORDIC flag to clear This parameter can be one of the following values:
  - CORDIC_FLAG_RRDY Result Ready Flag

Return value:
- None

__HAL_CORDIC_GET_IT_SOURCE

Description:
- Check whether the specified CORDIC interrupt is enabled or not.

Parameters:
- __HANDLE__: CORDIC handle.
- __INTERRUPT__: CORDIC interrupt to check This parameter can be one of the following values:
  - CORDIC_IT_IEN Enable Interrupt

Return value:
- FlagStatus

CORDIC status flags

CORDIC_FLAG_RRDY

Result Ready Flag

CORDIC Function

CORDIC_FUNCTION_COSINE

Cosine

CORDIC_FUNCTION_SINE

Sine

CORDIC_FUNCTION_PHASE

Phase

CORDIC_FUNCTION_MODULUS

Modulus

CORDIC_FUNCTION_ARCTANGENT

Arctangent

CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

CORDIC_FUNCTION_HSINE

Hyperbolic Sine

CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

CORDIC_FUNCTION_SQUAREROOT

Square Root

CORDIC Interrupts Enable bit

CORDIC_IT_IEN

Result ready interrupt enable

CORDIC input data size

CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

CORDIC Number of 32-bit read required after one calculation

CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data output (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

CORDIC Number of 32-bit write required for one calculation

CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

CORDIC Results Size

CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

CORDIC Precision in Cycles Number

CORDIC_PRECISION_1CYCLE

CORDIC_PRECISION_2CYCLES

CORDIC_PRECISION_3CYCLES

CORDIC_PRECISION_4CYCLES

CORDIC_PRECISION_5CYCLES

CORDIC_PRECISION_6CYCLES

CORDIC_PRECISION_7CYCLES

CORDIC_PRECISION_8CYCLES

CORDIC_PRECISION_9CYCLES

CORDIC_PRECISION_10CYCLES

CORDIC_PRECISION_11CYCLES

CORDIC_PRECISION_12CYCLES

CORDIC_PRECISION_13CYCLES

CORDIC_PRECISION_14CYCLES

CORDIC_PRECISION_15CYCLES

CORDIC Scaling factor

CORDIC_SCALE_0

CORDIC_SCALE_1

CORDIC_SCALE_2

CORDIC_SCALE_3

CORDIC_SCALE_4

CORDIC_SCALE_5

CORDIC_SCALE_6

CORDIC_SCALE_7

11 HAL CORTEX Generic Driver

11.1 CORTEX Firmware driver registers structures

11.1.1 MPU_Region_InitTypeDef

MPU_Region_InitTypeDef is defined in the stm32u5xx_hal_cortex.h

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint32_t LimitAddress***
- ***uint8_t AttributesIndex***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t IsShareable***

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
  Specifies the status of the region. This parameter can be a value of ***CORTEX_MPU_Region_Enable***
- ***uint8_t MPU_Region_InitTypeDef::Number***
  Specifies the index of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Number***
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
  Specifies the base address of the region to protect.
- ***uint32_t MPU_Region_InitTypeDef::LimitAddress***
  Specifies the limit address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::AttributesIndex***
  Specifies the memory attributes index. This parameter can be a value of ***CORTEX_MPU_Attributes_Number***
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
  Specifies the region access permission type. This parameter can be a value of ***CORTEX_MPU_Region_Permission_Attributes***
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
  Specifies the instruction access status. This parameter can be a value of ***CORTEX_MPU_Instruction_Access***
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
  Specifies the shareability status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Shareable***

11.1.2 MPU_Attributes_InitTypeDef

MPU_Attributes_InitTypeDef is defined in the stm32u5xx_hal_cortex.h

Data Fields

- ***uint8_t Number***
- ***uint8_t Attributes***

Field Documentation

- ***uint8_t MPU_Attributes_InitTypeDef::Number***
  Specifies the number of the memory attributes to configure. This parameter can be a value of ***CORTEX_MPU_Attributes_Number***
- ***uint8_t MPU_Attributes_InitTypeDef::Attributes***
  Specifies the memory attributes value. Attributes This parameter can be a combination of ***CORTEX_MPU_Attributes***

11.2 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

11.2.1 How to use this driver

How to configure Interrupts using CORTEX HAL driver

This section provides functions allowing to configure the NVIC interrupts (IRQ). The Cortex-M33 exceptions are managed by CMSIS functions.

1. Configure the NVIC Priority Grouping using HAL_NVIC_SetPriorityGrouping() function.
2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority().
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ().

Note: When the NVIC_PRIORITYGROUP_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the sub priority.

Note: IRQ priority order (sorted by highest to lowest priority):

- *Lowest pre-emption priority*
- *Lowest sub priority*
- *Lowest hardware priority (IRQ number)*

How to configure SysTick using CORTEX HAL driver

Setup SysTick Timer for time base.

- The HAL_SYSTICK_Config() function calls the SysTick_Config() function which is a CMSIS function that:
  - Configures the SysTick Reload register with value passed as function parameter.
  - Configures the SysTick IRQ priority to the lowest value (0x0F).
  - Resets the SysTick Counter register.
  - Configures the SysTick Counter clock source to be Core Clock Source (HCLK).
  - Enables the SysTick Interrupt.
  - Starts the SysTick Counter.
- You can change the SysTick Clock source to be HCLK_Div8 by calling the macro __HAL_CORTEX_SYSTICKCLK_CONFIG(SYSTICK_CLKSOURCE_HCLK_DIV8) just after the HAL_SYSTICK_Config() function call. The __HAL_CORTEX_SYSTICKCLK_CONFIG() macro is defined inside the stm32u5xx_hal_cortex.h file.
- You can change the SysTick IRQ priority by calling the HAL_NVIC_SetPriority(SysTick_IRQn,...) function just after the HAL_SYSTICK_Config() function call. The HAL_NVIC_SetPriority() call the NVIC_SetPriority() function which is a CMSIS function.
- To adjust the SysTick time base, use the following formula: Reload Value = SysTick Counter Clock (Hz) x Desired Time base (s)
  - Reload Value is the parameter to be passed for HAL_SYSTICK_Config() function
  - Reload Value should not exceed 0xFFFFFF

How to configure MPU (secure and non secure) using CORTEX HAL driver

This section provides functions allowing to Enable and configure the MPU secure and non-secure.

1. Enable the MPU using HAL_MPU_Enable() function.
2. Disable the MPU using HAL_MPU_Disable() function.
3. Enable the MPU using HAL_MPU_Enable_NS() function to address the non secure MPU.
4. Disable the MPU using HAL_MPU_Disable_NS() function to address the non secure MPU.
5. Configure the MPU region using HAL_MPU_ConfigRegion() and HAL_MPU_ConfigRegion_NS() to address the non secure MPU.
6. Configure the MPU Memory attributes using HAL_MPU_ConfigMemoryAttributes() and HAL_MPU_ConfigMemoryAttributes_NS() to address the non secure MPU.

11.2.2 Initialization and Configuration functions

This section provides the CORTEX HAL driver functions allowing to configure Interrupts SysTick functionalities

This section contains the following APIs:

- *HAL_NVIC_SetPriorityGrouping()*
- *HAL_NVIC_SetPriority()*
- *HAL_NVIC_EnableIRQ()*
- *HAL_NVIC_DisableIRQ()*
- *HAL_NVIC_SystemReset()*
- *HAL_SYSTICK_Config()*

11.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK, MPU) functionalities.

This section contains the following APIs:

- *HAL_NVIC_GetPriorityGrouping()*
- *HAL_NVIC_GetPriority()*
- *HAL_NVIC_SetPendingIRQ()*
- *HAL_NVIC_GetPendingIRQ()*
- *HAL_NVIC_ClearPendingIRQ()*
- *HAL_NVIC_GetActive()*
- *HAL_SYSTICK_CLKSourceConfig()*
- *HAL_SYSTICK_GetCLKSourceConfig()*
- *HAL_SYSTICK_IRQHandler()*
- *HAL_SYSTICK_Callback()*
- *HAL_MPU_Enable()*
- *HAL_MPU_Enable_NS()*
- *HAL_MPU_Disable()*
- *HAL_MPU_Disable_NS()*
- *HAL_MPU_EnableRegion()*
- *HAL_MPU_EnableRegion_NS()*
- *HAL_MPU_DisableRegion()*
- *HAL_MPU_DisableRegion_NS()*
- *HAL_MPU_ConfigRegion()*
- *HAL_MPU_ConfigRegion_NS()*
- *HAL_MPU_ConfigMemoryAttributes()*
- *HAL_MPU_ConfigMemoryAttributes_NS()*

11.2.4 Detailed description of functions

HAL_NVIC_SetPriorityGrouping

Function name

void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)

Function description

Set the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.

Parameters

- **PriorityGroup:** The priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority

Return values

- **None:**

Notes

- When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority.

HAL_NVIC_SetPriority

Function name

void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)

Function description

Set the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))
- **PreemptPriority:** The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority.

Return values

- **None:**

HAL_NVIC_EnableIRQ

Function name

void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)

Function description

Enable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))

Return values

- **None:**

Notes

- To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before.

HAL_NVIC_DisableIRQ

Function name

void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)

Function description

Disable a device specific interrupt in the NVIC interrupt controller.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))

Return values

- **None:**

HAL_NVIC_SystemReset

Function name

void HAL_NVIC_SystemReset (void)

Function description

Initiate a system reset request to reset the MCU.

Return values

- **None:**

HAL_SYSTICK_Config

Function name

uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)

Function description

Initialize the System Timer with interrupt enabled and start the System Tick Timer (SysTick): Counter is in free running mode to generate periodic interrupts.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- **status:** - 0 Function succeeded.
  - 1 Function failed.

HAL_NVIC_GetPriorityGrouping

Function name

uint32_t HAL_NVIC_GetPriorityGrouping (void)

Function description

Get the priority grouping field from the NVIC Interrupt Controller.

Return values

- **Priority:** grouping field (SCB->AIRCR [10:8] PRIGROUP field)

HAL_NVIC_GetPriority

Function name

void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t *const pPreemptPriority, uint32_t *const pSubPriority)

Function description

Get the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
  - NVIC_PRIORITYGROUP_0: 0 bit for pre-emption priority, 4 bits for subpriority
  - NVIC_PRIORITYGROUP_1: 1 bit for pre-emption priority, 3 bits for subpriority
  - NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority, 2 bits for subpriority
  - NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority, 1 bit for subpriority
  - NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority, 0 bit for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- **None:**

HAL_NVIC_GetPendingIRQ

Function name

uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)

Function description

Get Pending Interrupt (read the pending register in the NVIC and return the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  - 1 Interrupt status is pending.

HAL_NVIC_SetPendingIRQ

Function name

void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)

Function description

Set Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))

Return values

- **None:**

HAL_NVIC_ClearPendingIRQ

Function name

void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

Function description

Clear the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number. This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))

Return values

- **None:**

HAL_NVIC_GetActive

Function name

uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)

Function description

Get active interrupt (read the active register in NVIC and return the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32u5xxxx.h))

Return values

- **status:** - 0 Interrupt status is not pending.
  – 1 Interrupt status is pending.

HAL_SYSTICK_CLKSourceConfig

Function name

void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

Function description

Configure the SysTick clock source.

Parameters

- **CLKSource:** specifies the SysTick clock source. This parameter can be one of the following values:
  – SYSTICK_CLKSOURCE_LSI: LSI clock selected as SysTick clock source.
  – SYSTICK_CLKSOURCE_LSE: LSE clock selected as SysTick clock source.
  – SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
  – SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.

Return values

- **None:**

HAL_SYSTICK_GetCLKSourceConfig

Function name

uint32_t HAL_SYSTICK_GetCLKSourceConfig (void)

Function description

Get the SysTick clock source configuration.

Return values

- **SysTick:** clock source that can be one of the following values:
  - SYSTICK_CLKSOURCE_LSI: LSI clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_LSE: LSE clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.

HAL_SYSTICK_IRQHandler

Function name

void HAL_SYSTICK_IRQHandler (void)

Function description

Handle SYSTICK interrupt request.

Return values

- **None:**

HAL_SYSTICK_Callback

Function name

void HAL_SYSTICK_Callback (void)

Function description

SYSTICK callback.

Return values

- **None:**

HAL_MPU_Enable

Function name

void HAL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable the MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values

- **None:**

HAL_MPU_Disable

Function name

void HAL_MPU_Disable (void)

Function description

Disable the MPU.

Return values

- **None:**

HAL_MPU_EnableRegion

Function name

void HAL_MPU_EnableRegion (uint32_t RegionNumber)

Function description

Enable the MPU Region.

Parameters

- **RegionNumber:** Specifies the index of the region to enable. this parameter can be a value of CORTEX MPU Region Number

Return values

- **None:**

HAL_MPU_DisableRegion

Function name

void HAL_MPU_DisableRegion (uint32_t RegionNumber)

Function description

Disable the MPU Region.

Parameters

- **RegionNumber:** Specifies the index of the region to disable. this parameter can be a value of CORTEX MPU Region Number

Return values

- **None:**

HAL_MPU_ConfigRegion

Function name

void HAL_MPU_ConfigRegion (const MPU_Region_InitTypeDef *const pMPU_RegionInit)

Function description

Initialize and configure the Region and the memory to be protected.

Parameters

- **pMPU_RegionInit:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_ConfigMemoryAttributes

Function name

void HAL_MPU_ConfigMemoryAttributes (const MPU_Attributes_InitTypeDef *const pMPU_AttributesInit)

Function description

Initialize and configure the memory attributes.

Parameters

- **pMPU_AttributesInit:** Pointer to a MPU_Attributes_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_Enable_NS

Function name

void HAL_MPU_Enable_NS (uint32_t MPU_Control)

Function description

Enable the non-secure MPU.

Parameters

- **MPU_Control:** Specifies the control mode of the MPU during hard fault, NMI, FAULTMASK and privileged access to the default memory This parameter can be one of the following values:
  - MPU_HFNMI_PRIVDEF_NONE
  - MPU_HARDFAULT_NMI
  - MPU_PRIVILEGED_DEFAULT
  - MPU_HFNMI_PRIVDEF

Return values

- **None:**

HAL_MPU_Disable_NS

Function name

void HAL_MPU_Disable_NS (void)

Function description

Disable the non-secure MPU.

Return values

- **None:**

HAL_MPU_EnableRegion_NS

Function name

void HAL_MPU_EnableRegion_NS (uint32_t RegionNumber)

Function description

Enable the non-secure MPU Region.

Parameters

- **RegionNumber:** Specifies the index of the region to enable. this parameter can be a value of CORTEX MPU Region Number

Return values

- **None:**

HAL_MPU_DisableRegion_NS

Function name

void HAL_MPU_DisableRegion_NS (uint32_t RegionNumber)

Function description

Disable the non-secure MPU Region.

Parameters

- **RegionNumber:** Specifies the index of the region to disable. this parameter can be a value of CORTEX MPU Region Number

Return values

- **None:**

HAL_MPU_ConfigRegion_NS

Function name

void HAL_MPU_ConfigRegion_NS (const MPU_Region_InitTypeDef *const pMPU_RegionInit)

Function description

Initialize and configure the Region and the memory to be protected for non-secure MPU.

Parameters

- **pMPU_RegionInit:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

HAL_MPU_ConfigMemoryAttributes_NS

Function name

void HAL_MPU_ConfigMemoryAttributes_NS (const MPU_Attributes_InitTypeDef *const pMPU_AttributesInit)

Function description

Initialize and configure the memory attributes for non-secure MPU.

Parameters

- **pMPU_AttributesInit:** Pointer to a MPU_Attributes_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

11.3 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

11.3.1 CORTEX

CORTEX

CORTEX Exported Macros

OUTER

INNER_OUTER

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_NOT_SHAREABLE

MPU_ACCESS_OUTER_SHAREABLE

MPU_ACCESS_INNER_SHAREABLE

CORTEX MPU Attributes

MPU_DEVICE_NGNRNE

MPU_DEVICE_NGNRE

MPU_DEVICE_NGRE

MPU_DEVICE_GRE

MPU_WRITE_THROUGH

MPU_NOT_CACHEABLE

MPU_WRITE_BACK

MPU_TRANSIENT

MPU_NON_TRANSIENT

MPU_NO_ALLOCATE

MPU_W_ALLOCATE

MPU_R_ALLOCATE

MPU_RW_ALLOCATE

CORTEX MPU Memory Attributes Number

MPU_ATTRIBUTES_NUMBER0

MPU_ATTRIBUTES_NUMBER1

MPU_ATTRIBUTES_NUMBER2

MPU_ATTRIBUTES_NUMBER3

MPU_ATTRIBUTES_NUMBER4

MPU_ATTRIBUTES_NUMBER5

MPU_ATTRIBUTES_NUMBER6

MPU_ATTRIBUTES_NUMBER7

CORTEX MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_PRIV_RW

MPU_REGION_ALL_RW

MPU_REGION_PRIV_RO

MPU_REGION_ALL_RO

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0

0 bit for pre-emption priority, 4 bits for subpriority

NVIC_PRIORITYGROUP_1

1 bit for pre-emption priority, 3 bits for subpriority

NVIC_PRIORITYGROUP_2

2 bits for pre-emption priority, 2 bits for subpriority

NVIC_PRIORITYGROUP_3

3 bits for pre-emption priority, 1 bit for subpriority

NVIC_PRIORITYGROUP_4

4 bits for pre-emption priority, 0 bit for subpriority

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source

SYSTICK_CLKSOURCE_LSI

LSI clock selected as SysTick clock source

SYSTICK_CLKSOURCE_LSE

LSE clock selected as SysTick clock source

SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source

12 HAL CRC Generic Driver

12.1 CRC Firmware driver registers structures

12.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32u5xx_hal_crc.h

Data Fields

- ***uint8_t DefaultPolynomialUse***
- ***uint8_t DefaultInitValueUse***
- ***uint32_t GeneratingPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t InitValue***
- ***uint32_t InputDataInversionMode***
- ***uint32_t OutputDataInversionMode***

Field Documentation

- ***uint8_t CRC_InitTypeDef::DefaultPolynomialUse***
  This parameter is a value of ***CRC_Default_Polynomial*** and indicates if default polynomial is used. If set to DEFAULT_POLYNOMIAL_ENABLE, resort to default X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1. In that case, there is no need to set GeneratingPolynomial field. If otherwise set to DEFAULT_POLYNOMIAL_DISABLE, GeneratingPolynomial and CRCLength fields must be set.
- ***uint8_t CRC_InitTypeDef::DefaultInitValueUse***
  This parameter is a value of ***CRC_Default_InitValue_Use*** and indicates if default init value is used. If set to DEFAULT_INIT_VALUE_ENABLE, resort to default 0xFFFFFFFF value. In that case, there is no need to set InitValue field. If otherwise set to DEFAULT_INIT_VALUE_DISABLE, InitValue field must be set.
- ***uint32_t CRC_InitTypeDef::GeneratingPolynomial***
  Set CRC generating polynomial as a 7, 8, 16 or 32-bit long value for a polynomial degree respectively equal to 7, 8, 16 or 32. This field is written in normal, representation e.g., for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65. No need to specify it if DefaultPolynomialUse is set to DEFAULT_POLYNOMIAL_ENABLE.
- ***uint32_t CRC_InitTypeDef::CRCLength***
  This parameter is a value of ***CRC_Polynomial_Sizes*** and indicates CRC length. Value can be either one of
  - **CRC_POLYLENGTH_32B** (32-bit CRC),
  - **CRC_POLYLENGTH_16B** (16-bit CRC),
  - **CRC_POLYLENGTH_8B** (8-bit CRC),
  - **CRC_POLYLENGTH_7B** (7-bit CRC).
- ***uint32_t CRC_InitTypeDef::InitValue***
  Init value to initiate CRC computation. No need to specify it if DefaultInitValueUse is set to DEFAULT_INIT_VALUE_ENABLE.
- ***uint32_t CRC_InitTypeDef::InputDataInversionMode***
  This parameter is a value of ***CRCEx_Input_Data_Inversion*** and specifies input data inversion mode. Can be either one of the following values
  - **CRC_INPUTDATA_INVERSION_NONE** no input data inversion
  - **CRC_INPUTDATA_INVERSION_BYTE** byte-wise inversion, 0x1A2B3C4D becomes 0x58D43CB2
  - **CRC_INPUTDATA_INVERSION_HALFWORD** halfword-wise inversion, 0x1A2B3C4D becomes 0xD458B23C
  - **CRC_INPUTDATA_INVERSION_WORD** word-wise inversion, 0x1A2B3C4D becomes 0xB23CD458

- ***uint32_t CRC_InitTypeDef::OutputDataInversionMode***
  This parameter is a value of ***CRCEx_Output_Data_Inversion*** and specifies output data (i.e. CRC) inversion mode. Can be either
  – **CRC_OUTPUTDATA_INVERSION_DISABLE** no CRC inversion,
  – **CRC_OUTPUTDATA_INVERSION_ENABLE** CRC 0x11223344 is converted into 0x22CC4488

12.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32u5xx_hal_crc.h

Data Fields

- ***CRC_TypeDef * Instance***
- ***CRC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRC_StateTypeDef State***
- ***uint32_t InputDataFormat***

Field Documentation

- ***CRC_TypeDef* CRC_HandleTypeDef::Instance***
  Register base address
- ***CRC_InitTypeDef CRC_HandleTypeDef::Init***
  CRC configuration parameters
- ***HAL_LockTypeDef CRC_HandleTypeDef::Lock***
  CRC Locking object
- ***__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State***
  CRC communication state
- ***uint32_t CRC_HandleTypeDef::InputDataFormat***
  This parameter is a value of ***CRC_Input_Buffer_Format*** and specifies input data format. Can be either
  – **CRC_INPUTDATA_FORMAT_BYTES** input data is a stream of bytes (8-bit data)
  – **CRC_INPUTDATA_FORMAT_HALFWORDS** input data is a stream of half-words (16-bit data)
  – **CRC_INPUTDATA_FORMAT_WORDS** input data is a stream of words (32-bit data)

  Note that constant CRC_INPUT_FORMAT_UNDEFINED is defined but an initialization error must occur if InputBufferFormat is not one of the three values listed above

12.2 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

12.2.1 How to use this driver

- Enable CRC AHB clock using __HAL_RCC_CRC_CLK_ENABLE();
- Initialize CRC calculator
  – specify generating polynomial (peripheral default or non-default one)
  – specify initialization value (peripheral default or non-default one)
  – specify input data format
  – specify input or output data inversion mode if any
- Use HAL_CRC_Accumulate() function to compute the CRC value of the input data buffer starting with the previously computed CRC as initialization value
- Use HAL_CRC_Calculate() function to compute the CRC value of the input data buffer starting with the defined initialization value (default or non-default) to initiate CRC calculation

12.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle
- DeInitialize the CRC peripheral

- Initialize the CRC MSP (MCU Specific Package)
- DeInitialize the CRC MSP

This section contains the following APIs:

- *HAL_CRC_Init()*
- *HAL_CRC_DeInit()*
- *HAL_CRC_MspInit()*
- *HAL_CRC_MspDeInit()*

12.2.3 Peripheral Control functions

This section provides functions allowing to:

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer using combination of the previous CRC value and the new one.

or

- compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer independently of the previous CRC value.

This section contains the following APIs:

- *HAL_CRC_Accumulate()*
- *HAL_CRC_Calculate()*

12.2.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_CRC_GetState()*

12.2.5 Detailed description of functions

HAL_CRC_Init

Function name

HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)

Function description

Initialize the CRC according to the specified parameters in the CRC_InitTypeDef and create the associated handle.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_DeInit

Function name

HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC peripheral.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** status

HAL_CRC_MspInit

Function name

void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)

Function description

Initializes the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_MspDeInit

Function name

void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)

Function description

DeInitialize the CRC MSP.

Parameters

- **hcrc:** CRC handle

Return values

- **None:**

HAL_CRC_Accumulate

Function name

uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with the previously computed CRC as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_Calculate

Function name

uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)

Function description

Compute the 7, 8, 16 or 32-bit CRC value of an 8, 16 or 32-bit data buffer starting with hcrc->Instance->INIT as initialization value.

Parameters

- **hcrc:** CRC handle
- **pBuffer:** pointer to the input data buffer, exact input data format is provided by hcrc->InputDataFormat.
- **BufferLength:** input data buffer length (number of bytes if pBuffer type is * uint8_t, number of half-words if pBuffer type is * uint16_t, number of words if pBuffer type is * uint32_t).

Return values

- **uint32_t:** CRC (returned value LSBs for CRC shorter than 32 bits)

Notes

- By default, the API expects a uint32_t pointer as input buffer parameter. Input buffer pointers with other types simply need to be cast in uint32_t and the API will internally adjust its input data processing based on the handle field hcrc->InputDataFormat.

HAL_CRC_GetState

Function name

HAL_CRC_StateTypeDef HAL_CRC_GetState (const CRC_HandleTypeDef * hcrc)

Function description

Return the CRC handle state.

Parameters

- **hcrc:** CRC handle

Return values

- **HAL:** state

12.3 CRC Firmware driver defines

The following section lists the various define and macros of the module.

12.3.1 CRC

CRC

Default CRC computation initialization value

DEFAULT_CRC_INITVALUE

Initial CRC default value

Indicates whether or not default init value is used

DEFAULT_INIT_VALUE_ENABLE

Enable initial CRC default value

DEFAULT_INIT_VALUE_DISABLE

Disable initial CRC default value

Indicates whether or not default polynomial is used

DEFAULT_POLYNOMIAL_ENABLE

Enable default generating polynomial 0x04C11DB7

DEFAULT_POLYNOMIAL_DISABLE

Disable default generating polynomial 0x04C11DB7

Default CRC generating polynomial

DEFAULT_CRC32_POLY

X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 +X^8 + X^7 + X^5 + X^4 + X^2+ X +1

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE

Description:

- Reset CRC handle state.

Parameters:

- __HANDLE__: CRC handle.

Return value:

- None

__HAL_CRC_DR_RESET

Description:

- Reset CRC Data Register.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_INITIALCRCVALUE_CONFIG

Description:

- Set CRC INIT non-default value.

Parameters:

- __HANDLE__: CRC handle
- __INIT__: 32-bit initial value

Return value:

- None

__HAL_CRC_SET_IDR

Description:

- Store data in the Independent Data (ID) register.

Parameters:

- __HANDLE__: CRC handle
- __VALUE__: Value to be stored in the ID register

Return value:

- None

Notes:

- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

__HAL_CRC_GET_IDR

Description:
- Return the data stored in the Independent Data (ID) register.

Parameters:
- __HANDLE__: CRC handle

Return value:
- Value: of the ID register

Notes:
- Refer to the Reference Manual to get the authorized __VALUE__ length in bits

Input Buffer Format

CRC_INPUTDATA_FORMAT_UNDEFINED

Undefined input data format

CRC_INPUTDATA_FORMAT_BYTES

Input data in byte format

CRC_INPUTDATA_FORMAT_HALFWORDS

Input data in half-word format

CRC_INPUTDATA_FORMAT_WORDS

Input data in word format

Polynomial sizes to configure the peripheral

CRC_POLYLENGTH_32B

Resort to a 32-bit long generating polynomial

CRC_POLYLENGTH_16B

Resort to a 16-bit long generating polynomial

CRC_POLYLENGTH_8B

Resort to a 8-bit long generating polynomial

CRC_POLYLENGTH_7B

Resort to a 7-bit long generating polynomial

CRC polynomial possible sizes actual definitions

HAL_CRC_LENGTH_32B

32-bit long CRC

HAL_CRC_LENGTH_16B

16-bit long CRC

HAL_CRC_LENGTH_8B

8-bit long CRC

HAL_CRC_LENGTH_7B

7-bit long CRC

13 HAL CRC Extension Driver

13.1 CRCEx Firmware driver API description

The following section lists the various functions of the CRCEx library.

13.1.1 How to use this driver

- Set user-defined generating polynomial through HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

13.1.2 Extended configuration functions

This section provides functions allowing to:

- Configure the generating polynomial
- Configure the input data inversion
- Configure the output data inversion

This section contains the following APIs:

- *HAL_CRCEx_Polynomial_Set()*
- *HAL_CRCEx_Input_Data_Reverse()*
- *HAL_CRCEx_Output_Data_Reverse()*

13.1.3 Detailed description of functions

HAL_CRCEx_Polynomial_Set

Function name

HAL_StatusTypeDef HAL_CRCEx_Polynomial_Set (CRC_HandleTypeDef * hcrc, uint32_t Pol, uint32_t PolyLength)

Function description

Initialize the CRC polynomial if different from default one.

Parameters

- **hcrc:** CRC handle
- **Pol:** CRC generating polynomial (7, 8, 16 or 32-bit long). This parameter is written in normal representation, e.g.
  – for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65
  – for a polynomial of degree 16, X^16 + X^12 + X^5 + 1 is written 0x1021
- **PolyLength:** CRC polynomial length. This parameter can be one of the following values:
  – CRC_POLYLENGTH_7B 7-bit long CRC (generating polynomial of degree 7)
  – CRC_POLYLENGTH_8B 8-bit long CRC (generating polynomial of degree 8)
  – CRC_POLYLENGTH_16B 16-bit long CRC (generating polynomial of degree 16)
  – CRC_POLYLENGTH_32B 32-bit long CRC (generating polynomial of degree 32)

Return values

- **HAL:** status

HAL_CRCEx_Input_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Input_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t InputReverseMode)

Function description

Set the Reverse Input data mode.

Parameters

- **hcrc:** CRC handle
- **InputReverseMode:** Input Data inversion mode. This parameter can be one of the following values:
  - CRC_INPUTDATA_INVERSION_NONE no change in bit order (default value)
  - CRC_INPUTDATA_INVERSION_BYTE Byte-wise bit reversal
  - CRC_INPUTDATA_INVERSION_HALFWORD HalfWord-wise bit reversal
  - CRC_INPUTDATA_INVERSION_WORD Word-wise bit reversal

Return values

- **HAL:** status

HAL_CRCEx_Output_Data_Reverse

Function name

HAL_StatusTypeDef HAL_CRCEx_Output_Data_Reverse (CRC_HandleTypeDef * hcrc, uint32_t OutputReverseMode)

Function description

Set the Reverse Output data mode.

Parameters

- **hcrc:** CRC handle
- **OutputReverseMode:** Output Data inversion mode. This parameter can be one of the following values:
  - CRC_OUTPUTDATA_INVERSION_DISABLE no CRC inversion (default value)
  - CRC_OUTPUTDATA_INVERSION_ENABLE bit-level inversion (e.g. for a 8-bit CRC: 0xB5 becomes 0xAD)

Return values

- **HAL:** status

13.2 CRCEx Firmware driver defines

The following section lists the various define and macros of the module.

13.2.1 CRCEx

CRCEx

CRC Extended Exported Macros

__HAL_CRC_OUTPUTREVERSAL_ENABLE

Description:

- Set CRC output reversal.

Parameters:

- __HANDLE__: CRC handle

Return value:

- None

__HAL_CRC_OUTPUTREVERSAL_DISABLE

Description:
- Unset CRC output reversal.

Parameters:
- __HANDLE__: CRC handle

Return value:
- None

__HAL_CRC_POLYNOMIAL_CONFIG

Description:
- Set CRC non-default polynomial.

Parameters:
- __HANDLE__: CRC handle
- __POLYNOMIAL__: 7, 8, 16 or 32-bit polynomial

Return value:
- None

Input Data Inversion Modes

CRC_INPUTDATA_INVERSION_NONE

No input data inversion

CRC_INPUTDATA_INVERSION_BYTE

Byte-wise input data inversion

CRC_INPUTDATA_INVERSION_HALFWORD

HalfWord-wise input data inversion

CRC_INPUTDATA_INVERSION_WORD

Word-wise input data inversion

Output Data Inversion Modes

CRC_OUTPUTDATA_INVERSION_DISABLE

No output data inversion

CRC_OUTPUTDATA_INVERSION_ENABLE

Bit-wise output data inversion

14 HAL CRYP Generic Driver

14.1 CRYP Firmware driver registers structures

14.1.1 CRYP_ConfigTypeDef

CRYP_ConfigTypeDef is defined in the stm32u5xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t * Header***
- ***uint32_t HeaderSize***
- ***uint32_t * B0***
- ***uint32_t DataWidthUnit***
- ***uint32_t HeaderWidthUnit***
- ***uint32_t KeyIVConfigSkip***
- ***uint32_t KeyMode***
- ***uint32_t KeySelect***
- ***uint32_t KeyProtection***

Field Documentation

- ***uint32_t CRYP_ConfigTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ConfigTypeDef::KeySize***
  Used only in AES mode : 128, 192 or 256 bit key length in CRYP1. 128 or 256 bit key length in TinyAES This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ConfigTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ConfigTypeDef::pInitVect***
  The initialization vector used also as initialization counter in CTR mode
- ***uint32_t CRYP_ConfigTypeDef::Algorithm***
  DES/ TDES Algorithm ECB/CBC AES Algorithm ECB/CBC/CTR/GCM or CCM This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t* CRYP_ConfigTypeDef::Header***
  used only in AES GCM and CCM Algorithm for authentication, GCM : also known as Additional Authentication Data CCM : named B1 composed of the associated data length and Associated Data.
- ***uint32_t CRYP_ConfigTypeDef::HeaderSize***
  The size of header buffer
- ***uint32_t* CRYP_ConfigTypeDef::B0***
  B0 is first authentication block used only in AES CCM mode
- ***uint32_t CRYP_ConfigTypeDef::DataWidthUnit***
  Payload Data Width Unit, this parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::HeaderWidthUnit***
  Header Width Unit, this parameter can be value of ***CRYP_Header_Width_Unit***
- ***uint32_t CRYP_ConfigTypeDef::KeyIVConfigSkip***
  CRYP peripheral Key and IV configuration skip, to config Key and Initialization Vector only once and to skip configuration for consecutive processings. This parameter can be a value of ***CRYP_Configuration_Skip***
- ***uint32_t CRYP_ConfigTypeDef::KeyMode***
  Key mode selection, this parameter can be value of ***CRYP_Key_Mode***
- ***uint32_t CRYP_ConfigTypeDef::KeySelect***
  Only for SAES : Key selection, this parameter can be value of ***CRYP_Key_Select***

- ***uint32_t CRYP_ConfigTypeDef::KeyProtection***
  Only for SAES : Key protection, this parameter can be value of ***CRYP_Key_Protection***

14.1.2 CRYP_ContextTypeDef

CRYP_ContextTypeDef is defined in the stm32u5xx_hal_cryp.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint32_t * pKey***
- ***uint32_t * pInitVect***
- ***uint32_t Algorithm***
- ***uint32_t DataWidthUnit***
- ***uint32_t KeyIVConfigSkip***
- ***uint32_t KeyMode***
- ***uint32_t Phase***
- ***uint32_t KeyIVConfig***
- ***uint32_t CR_Reg***
- ***uint32_t IER_Reg***
- ***uint32_t IVR0_Reg***
- ***uint32_t IVR1_Reg***
- ***uint32_t IVR2_Reg***
- ***uint32_t IVR3_Reg***

Field Documentation

- ***uint32_t CRYP_ContextTypeDef::DataType***
  This parameter can be a value of ***CRYP_Data_Type***
- ***uint32_t CRYP_ContextTypeDef::KeySize***
  This parameter can be a value of ***CRYP_Key_Size***
- ***uint32_t* CRYP_ContextTypeDef::pKey***
  The key used for encryption/decryption
- ***uint32_t* CRYP_ContextTypeDef::pInitVect***
  The initialization vector, counter with CBC and CTR Algorithm
- ***uint32_t CRYP_ContextTypeDef::Algorithm***
  This parameter can be a value of ***CRYP_Algorithm_Mode***
- ***uint32_t CRYP_ContextTypeDef::DataWidthUnit***
  This parameter can be value of ***CRYP_Data_Width_Unit***
- ***uint32_t CRYP_ContextTypeDef::KeyIVConfigSkip***
  This parameter can be a value of ***CRYP_Configuration_Skip***
- ***uint32_t CRYP_ContextTypeDef::KeyMode***
  This parameter can be value of ***CRYP_Key_Mode***
- ***uint32_t CRYP_ContextTypeDef::Phase***
  CRYP peripheral phase
- ***uint32_t CRYP_ContextTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag
- ***uint32_t CRYP_ContextTypeDef::CR_Reg***
  CRYP CR register
- ***uint32_t CRYP_ContextTypeDef::IER_Reg***
  CRYP IER register
- ***uint32_t CRYP_ContextTypeDef::IVR0_Reg***
  CRYP IVR0 register
- ***uint32_t CRYP_ContextTypeDef::IVR1_Reg***
  CRYP IVR1 register
- ***uint32_t CRYP_ContextTypeDef::IVR2_Reg***
  CRYP IVR2 register

- ***uint32_t CRYP_ContextTypeDef::IVR3_Reg***
  CRYP IVR3 register

14.1.3 __CRYP_HandleTypeDef

__CRYP_HandleTypeDef is defined in the stm32u5xx_hal_cryp.h

Data Fields

- ***AES_TypeDef * Instance***
- ***CRYP_ConfigTypeDef Init***
- ***uint32_t * pCrypInBuffPtr***
- ***uint32_t * pCrypOutBuffPtr***
- ***__IO uint16_t CrypHeaderCount***
- ***__IO uint16_t CrypInCount***
- ***__IO uint16_t CrypOutCount***
- ***uint16_t Size***
- ***uint32_t Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_CRYP_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t KeyIVConfig***
- ***uint32_t SizesSum***
- ***void(* InCpltCallback***
- ***void(* OutCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***AES_TypeDef* __CRYP_HandleTypeDef::Instance***
  AES Register base address
- ***CRYP_ConfigTypeDef __CRYP_HandleTypeDef::Init***
  CRYP required parameters
- ***uint32_t* __CRYP_HandleTypeDef::pCrypInBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***uint32_t* __CRYP_HandleTypeDef::pCrypOutBuffPtr***
  Pointer to CRYP processing (encryption, decryption,...) buffer
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypHeaderCount***
  Counter of header data in words
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypInCount***
  Counter of input data in words
- ***__IO uint16_t __CRYP_HandleTypeDef::CrypOutCount***
  Counter of output data in words
- ***uint16_t __CRYP_HandleTypeDef::Size***
  length of input data in word or in byte, according to DataWidthUnit
- ***uint32_t __CRYP_HandleTypeDef::Phase***
  CRYP peripheral phase
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmain***
  CRYP In DMA handle parameters
- ***DMA_HandleTypeDef* __CRYP_HandleTypeDef::hdmaout***
  CRYP Out DMA handle parameters
- ***HAL_LockTypeDef __CRYP_HandleTypeDef::Lock***
  CRYP locking object

- ***__IO HAL_CRYP_STATETypeDef __CRYP_HandleTypeDef::State***
  CRYP peripheral state
- ***__IO uint32_t __CRYP_HandleTypeDef::ErrorCode***
  CRYP peripheral error code
- ***uint32_t __CRYP_HandleTypeDef::KeyIVConfig***
  CRYP peripheral Key and IV configuration flag, used when configuration can be skipped
- ***uint32_t __CRYP_HandleTypeDef::SizesSum***
  Sum of successive payloads lengths (in bytes), stored for a single signature computation after several messages processing
- ***void(* __CRYP_HandleTypeDef::InCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Input FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::OutCpltCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Output FIFO transfer completed callback
- ***void(* __CRYP_HandleTypeDef::ErrorCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Error callback
- ***void(* __CRYP_HandleTypeDef::MspInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp Init callback
- ***void(* __CRYP_HandleTypeDef::MspDeInitCallback)(struct __CRYP_HandleTypeDef *hcryp)***
  CRYP Msp DeInit callback

14.2 CRYP Firmware driver API description

The following section lists the various functions of the CRYP library.

14.2.1 How to use this driver

The CRYP HAL driver can be used as follows:

1. Initialize the CRYP low level resources by implementing the HAL_CRYP_MspInit():
   a. Enable the CRYP interface clock using __HAL_RCC_CRYP_CLK_ENABLE()or __HAL_RCC_AES_CLK_ENABLE
   b. In case of using interrupts (e.g. HAL_CRYP_Encrypt_IT())
      - Configure the CRYP interrupt priority using HAL_NVIC_SetPriority()
      - Enable the CRYP IRQ handler using HAL_NVIC_EnableIRQ()
      - In CRYP IRQ handler, call HAL_CRYP_IRQHandler()
   c. In case of using DMA to control data transfer (e.g. HAL_CRYP_Encrypt_DMA())
      - Enable the DMAx interface clock using __RCC_DMAx_CLK_ENABLE()
      - Configure and enable two DMA channels one for managing data transfer from memory to peripheral (input channel) and another channel for managing data transfer from peripheral to memory (output channel)
      - Associate the initialized DMA handle to the CRYP DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the two DMA channels. The output channel should have higher priority than the input channel HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()

2. Initialize the CRYP according to the specified parameters :
   a. The data type: 1-bit, 8-bit, 16-bit or 32-bit.
   b. The key size: 128, 192 or 256.
   c. The AES Algorithm ECB/CBC/CTR/GCM or CCM.
   d. The initialization vector (counter). It is not used in ECB mode.
   e. The key buffer used for encryption/decryption.
      - In some specific configurations, the key is written by the application code out of the HAL scope. In that case, user can still resort to the HAL APIs as usual but must make sure that pKey pointer is set to NULL.
   f. The DataWidthUnit field. It specifies whether the data length (or the payload length for authentication algorithms) is in words or bytes.
   g. The Header used only in AES GCM and CCM Algorithm for authentication.
   h. The HeaderSize used to give size of header buffer in word or bytes, depending upon HeaderWidthUnit field.
   i. The HeaderWidthUnit field. It specifies whether the header length (for authentication algorithms) is in words or bytes.
   j. The B0 block is the first authentication block used only in AES CCM mode.
   k. The KeyIVConfigSkip used to process several messages in a row.
   l. The KeyMode used to special key operation modes (for SAES : wrapped key, shared key with AES peripheral).
   m. The KeySelect, Only for SAES, used to select key from different key source.
   n. The KeyProtection, Only for SAES, used for security context enforcement.
3. Three processing (encryption/decryption) functions are available:
   a. Polling mode: encryption and decryption APIs are blocking functions i.e. they process the data and wait till the processing is finished, e.g. HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
   b. Interrupt mode: encryption and decryption APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
   c. DMA mode: encryption and decryption APIs are not blocking functions i.e. the data transfer is ensured by DMA, e.g. HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA
4. When the processing function is called at first time after HAL_CRYP_Init() the CRYP peripheral is configured and processes the buffer in input. At second call, no need to Initialize the CRYP, user have to get current configuration via HAL_CRYP_GetConfig() API, then only HAL_CRYP_SetConfig() is requested to set new parameters, finally user can start encryption/decryption.
5. Call HAL_CRYP_DeInit() to deinitialize the CRYP peripheral.
6. To process a single message with consecutive calls to HAL_CRYP_Encrypt() or HAL_CRYP_Decrypt() without having to configure again the Key or the Initialization Vector between each API call, the field KeyIVConfigSkip of the initialization structure must be set to CRYP_KEYIVCONFIG_ONCE. Same is true for consecutive calls of HAL_CRYP_Encrypt_IT(), HAL_CRYP_Decrypt_IT(), HAL_CRYP_Encrypt_DMA() or HAL_CRYP_Decrypt_DMA().

The cryptographic processor supports following standards:

1. The advanced encryption standard (AES) supported:
   a. 128-bit data block processing
   b. chaining modes supported :
      - Electronic Code Book(ECB)
      - Cipher Block Chaining (CBC)
      - Counter mode (CTR)
      - Galois/counter mode (GCM/GMAC)
      - Counter with Cipher Block Chaining-Message(CCM)
   c. keys length Supported : 128-bit and 256-bit

Note: *Specific care must be taken to format the key and the Initialization Vector IV!*

If the key is defined as a 128-bit long array key[127..0] = {b127 ... b0} where b127 is the MSB and b0 the LSB, the key must be stored in MCU memory

- as a sequence of words where the MSB word comes first (occupies the lowest memory address)
  - address n+0 : 0b b127 .. b120 b119 .. b112 b111 .. b104 b103 .. b96
  - address n+4 : 0b b95 .. b88 b87 .. b80 b79 .. b72 b71 .. b64
  - address n+8 : 0b b63 .. b56 b55 .. b48 b47 .. b40 b39 .. b32
  - address n+C : 0b b31 .. b24 b23 .. b16 b15 .. b8 b7 .. b0

Hereafter, another illustration when considering a 128-bit long key made of 16 bytes {B15..B0}. The 4 32-bit words that make the key must be stored as follows in MCU memory:

- address n+0 : 0x B15 B14 B13 B12
- address n+4 : 0x B11 B10 B9 B8
- address n+8 : 0x B7 B6 B5 B4
- address n+C : 0x B3 B2 B1 B0

which leads to the expected setting

- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Same format must be applied for a 256-bit long key made of 32 bytes {B31..B0}. The 8 32-bit words that make the key must be stored as follows in MCU memory:

- address n+00 : 0x B31 B30 B29 B28
- address n+04 : 0x B27 B26 B25 B24
- address n+08 : 0x B23 B22 B21 B20
- address n+0C : 0x B19 B18 B17 B16
- address n+10 : 0x B15 B14 B13 B12
- address n+14 : 0x B11 B10 B9 B8
- address n+18 : 0x B7 B6 B5 B4
- address n+1C : 0x B3 B2 B1 B0

which leads to the expected setting

- AES_KEYR7 = 0x B31 B30 B29 B28
- AES_KEYR6 = 0x B27 B26 B25 B24
- AES_KEYR5 = 0x B23 B22 B21 B20
- AES_KEYR4 = 0x B19 B18 B17 B16
- AES_KEYR3 = 0x B15 B14 B13 B12
- AES_KEYR2 = 0x B11 B10 B9 B8
- AES_KEYR1 = 0x B7 B6 B5 B4
- AES_KEYR0 = 0x B3 B2 B1 B0

Initialization Vector IV (4 32-bit words) format must follow the same as that of a 128-bit long key.

Note that key and IV registers are not sensitive to swap mode selection.

This section describes the AES Galois/counter mode (GCM) supported by the peripherals:

1. Algorithm supported :
   a. Galois/counter mode (GCM)
   b. Galois message authentication code (GMAC) :is exactly the same as GCM algorithm composed only by an header.
2. Four phases are performed in GCM :
   a. Init phase: peripheral prepares the GCM hash subkey (H) and do the IV processing
   b. Header phase: peripheral processes the Additional Authenticated Data (AAD), with hash computation only.
   c. Payload phase: peripheral processes the plaintext (P) with hash computation + keystream encryption + data XORing. It works in a similar way for ciphertext (C).
   d. Final phase: peripheral generates the authenticated tag (T) using the last block of data. HAL_CRYPEx_AESGCM_GenerateAuthTAG API used in this phase to generate 4 words which correspond to the Tag. user should consider only part of this 4 words, if Tag length is less than 128 bits.

3. structure of message construction in GCM is defined as below :
   a. 16 bytes Initial Counter Block (ICB)composed of IV and counter
   b. The authenticated header A (also knows as Additional Authentication Data AAD) this part of the message is only authenticated, not encrypted.
   c. The plaintext message P is both authenticated and encrypted as ciphertext. GCM standard specifies that ciphertext has same bit length as the plaintext.
   d. The last block is composed of the length of A (on 64 bits) and the length of ciphertext (on 64 bits)

A more detailed description of the GCM message structure is available below.

This section describe The AES Counter with Cipher Block Chaining-Message Authentication Code (CCM) supported by the peripheral:

1. Specific parameters for CCM :
   a. B0 block : follows NIST Special Publication 800-38C,
   b. B1 block (header)
   c. CTRx block : control blocks

A detailed description of the CCM message structure is available below.

1. CCM in peripheral:
   a. To perform message payload encryption or decryption AES is configured in CTR mode.
   b. For authentication two phases are performed : - Header phase: peripheral processes the Additional Authenticated Data (AAD) first, then the cleartext message only cleartext payload (not the ciphertext payload) is used and no output.
   c. Final phase: peripheral generates the authenticated tag (T) using the last block of data. HAL_CRYPEx_AESCCM_GenerateAuthTAG API used in this phase to generate 4 words which correspond to the Tag. user should consider only part of this 4 words, if Tag length is less than 128 bits

Callback registration

The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_CRYP_RegisterCallback() or HAL_CRYP_RegisterXXXCallback() to register an interrupt callback.

Function @ref HAL_CRYP_RegisterCallback() allows to register following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_CRYP_UnRegisterCallback() to reset a callback to the default weak function. @ref HAL_CRYP_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- InCpltCallback : Input FIFO transfer completed callback.
- OutCpltCallback : Output FIFO transfer completed callback.
- ErrorCallback : callback for error detection.
- MspInitCallback : CRYP MspInit.
- MspDeInitCallback : CRYP MspDeInit.

By default, after the @ref HAL_CRYP_Init() and when the state is HAL_CRYP_STATE_RESET all callbacks are set to the corresponding weak functions : examples @ref HAL_CRYP_InCpltCallback() , @ref HAL_CRYP_OutCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak function in the @ref HAL_CRYP_Init()/ @ref HAL_CRYP_DeInit() only when these callbacks are null (not registered beforehand). if not, MspInit or MspDeInit are not null, the @ref HAL_CRYP_Init() / @ref HAL_CRYP_DeInit() keep and use the user MspInit/MspDeInit functions (registered beforehand)

Callbacks can be registered/unregistered in HAL_CRYP_STATE_READY state only. Exception done MspInit/ MspDeInit callbacks that can be registered/unregistered in HAL_CRYP_STATE_READY or HAL_CRYP_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_CRYP_RegisterCallback() before calling @ref HAL_CRYP_DeInit() or @ref HAL_CRYP_Init() function.

When The compilation define USE_HAL_CRYP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Suspend/Resume feature

The compilation define USE_HAL_CRYP_SUSPEND_RESUME when set to 1 allows the user to resort to the suspend/resume feature. A low priority block processing can be suspended to process a high priority block instead. When the high priority block processing is over, the low priority block processing can be resumed, restarting from the point where it was suspended. This feature is applicable only in non-blocking interrupt mode.

User must resort to HAL_CRYP_Suspend() to suspend the low priority block processing. This API manages the hardware block processing suspension and saves all the internal data that will be needed to restart later on. Upon HAL_CRYP_Suspend() completion, the user can launch the processing of any other block (high priority block processing).

When the high priority block processing is over, user must invoke HAL_CRYP_Resume() to resume the low priority block processing. Ciphering (or deciphering) restarts from the suspension point and ends as usual.

HAL_CRYP_Suspend() reports an error when the suspension request is sent too late (i.e when the low priority block processing is about to end). There is no use to suspend the tag generation processing for authentication algorithms.

Note: If the key is written out of HAL scope (case pKey pointer set to NULL by the user), the block processing suspension/resumption mechanism is NOT applicable.

Note: If the Key and Initialization Vector are configured only once and configuration is skipped for consecutive processings (case KeyIVConfigSkip set to CRYP_KEYIVCONFIG_ONCE), the block processing suspension/ resumption mechanism is NOT applicable.

14.2.2 Initialization, de-initialization and Set and Get configuration functions

This section provides functions allowing to:

- Initialize the CRYP
- DeInitialize the CRYP
- Initialize the CRYP MSP
- DeInitialize the CRYP MSP
- configure CRYP (HAL_CRYP_SetConfig) with the specified parameters in the CRYP_ConfigTypeDef Parameters which are configured in This section are :
- Key size
- Data Type : 32,16, 8 or 1bit
- AlgoMode : ECB,CBC,CTR,GCM/GMAC and CCM in AES Standard.
- Get CRYP configuration (HAL_CRYP_GetConfig) from the specified parameters in the CRYP_HandleTypeDef
- For interleave mode, API HAL_CRYP_SaveContext and HAL_CRYP_RestoreContext to be used to save then Restore CRYP configuration and parameters. CRYP_IVCONFIG_ONCE should be selected for KeyIVConfigSkip parameter. Only polling mode is supported, interleave mode should be used with HAL_CRYP_Encrypt and HAL_CRYP_Decrypt API.

This section contains the following APIs:

- *HAL_CRYP_Init()*
- *HAL_CRYP_DeInit()*
- *HAL_CRYP_SetConfig()*
- *HAL_CRYP_GetConfig()*
- *HAL_CRYP_MspInit()*
- *HAL_CRYP_MspDeInit()*
- *HAL_CRYP_RegisterCallback()*
- *HAL_CRYP_UnRegisterCallback()*
- *HAL_CRYP_SaveContext()*
- *HAL_CRYP_RestoreContext()*

14.2.3 Encrypt Decrypt functions

This section provides API allowing to Encrypt/Decrypt Data following

- Standard AES algorithms supported by the peripheral: - Electronic Code Book(ECB) - Cipher Block Chaining (CBC) - Counter mode (CTR) - Cipher Block Chaining (CBC) - Counter mode (CTR) - Galois/counter mode (GCM) - Counter with Cipher Block Chaining-Message(CCM)

Three processing functions are available:

- Polling mode : HAL_CRYP_Encrypt & HAL_CRYP_Decrypt
- Interrupt mode : HAL_CRYP_Encrypt_IT & HAL_CRYP_Decrypt_IT
- DMA mode : HAL_CRYP_Encrypt_DMA & HAL_CRYP_Decrypt_DMA

This section contains the following APIs:

- *HAL_CRYP_Encrypt()*
- *HAL_CRYP_Decrypt()*
- *HAL_CRYP_Encrypt_IT()*
- *HAL_CRYP_Decrypt_IT()*
- *HAL_CRYP_Encrypt_DMA()*
- *HAL_CRYP_Decrypt_DMA()*

14.2.4 CRYP IRQ handler management

This section provides CRYP IRQ handler and callback functions.

- HAL_CRYP_IRQHandler CRYP interrupt request
- HAL_CRYP_InCpltCallback input data transfer complete callback
- HAL_CRYP_OutCpltCallback output data transfer complete callback
- HAL_CRYP_ErrorCallback CRYP error callback
- HAL_CRYP_GetState return the CRYP state
- HAL_CRYP_GetError return the CRYP error code

This section contains the following APIs:

- *HAL_CRYP_IRQHandler()*
- *HAL_CRYP_GetError()*
- *HAL_CRYP_GetState()*
- *HAL_CRYP_InCpltCallback()*
- *HAL_CRYP_OutCpltCallback()*
- *HAL_CRYP_ErrorCallback()*

14.2.5 Detailed description of functions

HAL_CRYP_Init

Function name

HAL_StatusTypeDef HAL_CRYP_Init (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP according to the specified parameters in the CRYP_ConfigTypeDef and creates the associated handle.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_DeInit

Function name

HAL_StatusTypeDef HAL_CRYP_DeInit (CRYP_HandleTypeDef * hcryp)

Function description

De-Initializes the CRYP peripheral.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_MspInit

Function name

void HAL_CRYP_MspInit (CRYP_HandleTypeDef * hcryp)

Function description

Initializes the CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_MspDeInit

Function name

void HAL_CRYP_MspDeInit (CRYP_HandleTypeDef * hcryp)

Function description

DeInitializes CRYP MSP.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_SetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_SetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Configure the CRYP according to the specified parameters in the CRYP_ConfigTypeDef.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure
- **pConf:** pointer to a CRYP_ConfigTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_GetConfig

Function name

HAL_StatusTypeDef HAL_CRYP_GetConfig (CRYP_HandleTypeDef * hcryp, CRYP_ConfigTypeDef * pConf)

Function description

Get CRYP Configuration parameters in associated handle.

Parameters

- **pConf:** pointer to a CRYP_ConfigTypeDef structure
- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **HAL:** status

HAL_CRYP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_CRYP_RegisterCallback (CRYP_HandleTypeDef * hcryp, HAL_CRYP_CallbackIDTypeDef CallbackID, pCRYP_CallbackTypeDef pCallback)

Function description

Register a User CRYP Callback To be used instead of the weak predefined callback.

Parameters

- **hcryp:** cryp handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_CRYP_INPUT_COMPLETE_CB_ID Input FIFO transfer completed callback ID
  - HAL_CRYP_OUTPUT_COMPLETE_CB_ID Output FIFO transfer completed callback ID
  - HAL_CRYP_ERROR_CB_ID Error callback ID
  - HAL_CRYP_MSPINIT_CB_ID MspInit callback ID
  - HAL_CRYP_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_CRYP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_CRYP_UnRegisterCallback (CRYP_HandleTypeDef * hcryp, HAL_CRYP_CallbackIDTypeDef CallbackID)

Function description

Unregister an CRYP Callback CRYP callback is redirected to the weak predefined callback.

Parameters

- **hcryp:** cryp handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_CRYP_INPUT_COMPLETE_CB_ID Input FIFO transfer completed callback ID
  - HAL_CRYP_OUTPUT_COMPLETE_CB_ID Output FIFO transfer completed callback ID
  - HAL_CRYP_ERROR_CB_ID Error callback ID
  - HAL_CRYP_MSPINIT_CB_ID MspInit callback ID
  - HAL_CRYP_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_CRYP_SaveContext

Function name

HAL_StatusTypeDef HAL_CRYP_SaveContext (CRYP_HandleTypeDef * hcryp, CRYP_ContextTypeDef * pcont)

Function description

CRYP peripheral parameters storage when processing Interleaved mode .

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pcont:** pointer to a CRYP_ContextTypeDef structure where CRYP parameters will be stored.

Return values

- **HAL:** status

HAL_CRYP_RestoreContext

Function name

HAL_StatusTypeDef HAL_CRYP_RestoreContext (CRYP_HandleTypeDef * hcryp, CRYP_ContextTypeDef * pcont)

Function description

Restore CRYP parameters needed for Interleaved mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pcont:** pointer to a CRYP_ContextTypeDef structure that contains CRYP parameters stored.

Return values

- **HAL:** status

HAL_CRYP_Encrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput, uint32_t Timeout)

Function description

Encryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (plaintext)
- **Size:** Length of the plaintext buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(ciphertext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Decrypt

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput, uint32_t Timeout)

Function description

Decryption mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(plaintext)
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYP_Encrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Encryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (plaintext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_IT

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_IT (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Decryption in interrupt mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_Encrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Encrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Encryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (plaintext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(ciphertext)

Return values

- **HAL:** status

HAL_CRYP_Decrypt_DMA

Function name

HAL_StatusTypeDef HAL_CRYP_Decrypt_DMA (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint16_t Size, uint32_t * pOutput)

Function description

Decryption in DMA mode.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the input buffer (ciphertext)
- **Size:** Length of the input buffer either in word or in byte, according to DataWidthUnit
- **pOutput:** Pointer to the output buffer(plaintext)

Return values

- **HAL:** status

HAL_CRYP_IRQHandler

Function name

void HAL_CRYP_IRQHandler (CRYP_HandleTypeDef * hcryp)

Function description

This function handles cryptographic interrupt request.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module

Return values

- **None:**

HAL_CRYP_GetState

Function name

HAL_CRYP_STATETypeDef HAL_CRYP_GetState (const CRYP_HandleTypeDef * hcryp)

Function description

Returns the CRYP state.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **HAL:** state

HAL_CRYP_InCpltCallback

Function name

void HAL_CRYP_InCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Input FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_OutCpltCallback

Function name

void HAL_CRYP_OutCpltCallback (CRYP_HandleTypeDef * hcryp)

Function description

Output FIFO transfer completed callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_ErrorCallback

Function name

void HAL_CRYP_ErrorCallback (CRYP_HandleTypeDef * hcryp)

Function description

CRYP error callback.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module.

Return values

- **None:**

HAL_CRYP_GetError

Function name

uint32_t HAL_CRYP_GetError (const CRYP_HandleTypeDef * hcryp)

Function description

Return the CRYP error code.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for the CRYP peripheral

Return values

- **CRYP:** error code

14.3 CRYP Firmware driver defines

The following section lists the various define and macros of the module.

14.3.1 CRYP

CRYP

CRYP Algorithm Mode

CRYP_AES_ECB

Electronic codebook chaining algorithm

CRYP_AES_CBC

Cipher block chaining algorithm

CRYP_AES_CTR

Counter mode chaining algorithm

CRYP_AES_GCM_GMAC

Galois counter mode - Galois message authentication code

CRYP_AES_CCM

Counter with Cipher Mode

CRYP CLEAR Flags

CRYP_CLEAR_CCF

Computation Complete Flag Clear

CRYP_CLEAR_RWEIF

Clear Error Flag : RWEIF in AES_ISR and both RDERR and WRERR flags in AES_SR

CRYP_CLEAR_KEIF

Clear Key Error Flag: KEIF in AES_ISR

CRYP_CLEAR_RNGEIF

Clear rng Error Flag: RNGEIF in AES_ISR

CRYP Key and IV Configuration Skip Mode

CRYP_KEYIVCONFIG_ALWAYS

Peripheral Key and IV configuration to do systematically

CRYP_KEYIVCONFIG_ONCE

Peripheral Key and IV configuration to do only once

CRYP_KEYNOCONFIG

Peripheral Key configuration to not do

CRYP_IVCONFIG_ONCE

Peripheral IV configuration do once for interleave mode

CRYP Data Type

CRYP_DATATYPE_32B

CRYP_DATATYPE_16B

CRYP_DATATYPE_8B

CRYP_DATATYPE_1B

CRYP_NO_SWAP

32-bit data type (no swapping)

CRYP_HALFWORD_SWAP

16-bit data type (half-word swapping)

CRYP_BYTE_SWAP

8-bit data type (byte swapping)

CRYP_BIT_SWAP

1-bit data type (bit swapping)

CRYP Data Width Unit

CRYP_DATAWIDTHUNIT_WORD

By default, size unit is word

CRYP_DATAWIDTHUNIT_BYTE

By default, size unit is byte

CRYP Error Definition

HAL_CRYP_ERROR_NONE

No error

HAL_CRYP_ERROR_WRITE
Write error

HAL_CRYP_ERROR_READ
Read error

HAL_CRYP_ERROR_DMA
DMA error

HAL_CRYP_ERROR_BUSY
Busy flag error

HAL_CRYP_ERROR_TIMEOUT
Timeout error

HAL_CRYP_ERROR_NOT_SUPPORTED
Not supported mode

HAL_CRYP_ERROR_AUTH_TAG_SEQUENCE
Sequence are not respected only for GCM or CCM

HAL_CRYP_ERROR_INVALID_CALLBACK
Invalid Callback error

HAL_CRYP_ERROR_KEY
Key error

HAL_CRYP_ERROR_RNG
Rng error

CRYP Exported Macros

__HAL_CRYP_RESET_HANDLE_STATE

Description:
- Reset CRYP handle state.

Parameters:
- __HANDLE__: specifies the CRYP handle.

Return value:
- None

__HAL_CRYP_ENABLE

Description:
- Enable/Disable the CRYP peripheral.

Parameters:
- __HANDLE__: specifies the CRYP handle.

Return value:
- None

__HAL_CRYP_DISABLE

__HAL_CRYP_GET_FLAG

Description:

- Check whether the specified CRYP status flag is set or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values for TinyAES:
  - CRYP_FLAG_KEYVALID Key valid flag
  - CRYP_FLAG_BUSY GCM process suspension forbidden or transferring a shared key from SAES IP.
  - CRYP_FLAG_WRERR Write Error flag
  - CRYP_FLAG_RDERR Read Error flag
  - CRYP_FLAG_CCF Computation Complete flag
  - CRYP_FLAG_KEIF Key error flag
  - CRYP_FLAG_RWEIF Read/write Error flag

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_CRYP_CLEAR_FLAG

Description:

- Clear the CRYP pending status flag.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - CRYP_CLEAR_RWEIF Read (RDERR), Write (WRERR) or Read/write (RWEIF) Error Flag Clear
  - CRYP_CLEAR_CCF Computation Complete Flag (CCF) Clear
  - CRYP_CLEAR_KEIF Key error interrupt flag clear

Return value:

- None

__HAL_CRYP_GET_IT_SOURCE

Description:

- Check whether the specified CRYP interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP interrupt source to check This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt

Return value:

- State: of interruption (TRUE or FALSE).

__HAL_CRYP_ENABLE_IT

Description:

- Enable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt

Return value:

- None

__HAL_CRYP_DISABLE_IT

Description:

- Disable the CRYP interrupt.

Parameters:

- __HANDLE__: specifies the CRYP handle.
- __INTERRUPT__: CRYP Interrupt. This parameter can be one of the following values for TinyAES:
  - CRYP_IT_RWEIE Error interrupt (used for RDERR and WRERR)
  - CRYP_IT_CCFIE Computation Complete interrupt
  - CRYP_IT_KEIE Key error interrupt

Return value:

- None

CRYP Flags

CRYP_FLAG_BUSY

GCM process suspension forbidden also set when transferring a shared key from SAES peripheral

CRYP_FLAG_WRERR

Write Error flag

CRYP_FLAG_RDERR

Read error flag

CRYP_FLAG_CCF

Computation completed flag as AES_ISR_CCF

CRYP_FLAG_KEYVALID

Key Valid flag

CRYP_FLAG_KEIF

Key error interrupt flag

CRYP_FLAG_RWEIF

Read or write error Interrupt flag

CRYP_FLAG_RNGEIF

RNG error interrupt flag

CRYP Header Width Unit

CRYP_HEADERWIDTHUNIT_WORD

By default, header size unit is word

CRYP_HEADERWIDTHUNIT_BYTE

By default, header size unit is byte

CRYP Interrupt

CRYP_IT_CCFIE

Computation Complete interrupt enable

CRYP_IT_RWEIE

Read or write Error interrupt enable

CRYP_IT_KEIE

Key error interrupt enable

CRYP_IT_RNGEIE

Rng error interrupt enable

CRYP Key Mode

CRYP_KEYMODE_NORMAL

Normal key usage, Key registers are freely usable

CRYP_KEYMODE_SHARED

Shared key

CRYP_KEYMODE_WRAPPED

Only for SAES, Wrapped key: to encrypt or decrypt AES keys

CRYP Key Protection

CRYP_KEYPROT_ENABLE

Only for SAES, Key protection between 2 applications with different security contexts

CRYP_KEYPROT_DISABLE

Only for SAES, Key not protected between 2 applications with different security contexts

CRYP Key Select

CRYP_KEYSEL_NORMAL

Normal key, key registers SAES_KEYx or CRYP_KEYx

CRYP_KEYSEL_HW

Only for SAES, Hardware key : derived hardware unique key (DHUK 256-bit)

CRYP_KEYSEL_SW

Only for SAES, Software key : boot hardware key BHK (256-bit)

CRYP_KEYSEL_HSW

Only for SAES, DHUK XOR BHK Hardware unique key XOR software key

CRYP Key Share ID

CRYP_KSHAREID_AES

Share SAES Key with AES peripheral

CRYP Key Size

CRYP_KEYSIZE_128B

128-bit long key

CRYP_KEYSIZE_256B

256-bit long key

15 HAL CRYP Extension Driver

15.1 CRYPEx Firmware driver API description

The following section lists the various functions of the CRYPEx library.

15.1.1 Extended AES processing functions

This section provides functions allowing to generate the authentication TAG in Polling mode

1. HAL_CRYPEx_AESGCM_GenerateAuthTAG
2. HAL_CRYPEx_AESCCM_GenerateAuthTAG they should be used after Encrypt/Decrypt operation.

This section contains the following APIs:

- *HAL_CRYPEx_AESGCM_GenerateAuthTAG()*
- *HAL_CRYPEx_AESCCM_GenerateAuthTAG()*

15.1.2 Wrap and Unwrap key

This section provides API allowing to wrap (encrypt) and unwrap (decrypt) key using one of the following keys, and AES Algorithm. Key selection : - Derived hardware unique key (DHUK) - XOR of DHUK and BHK - Boot hardware key (BHK)

This section contains the following APIs:

- *HAL_CRYPEx_WrapKey()*
- *HAL_CRYPEx_UnwrapKey()*

15.1.3 Encrypt and Decrypt Shared key functions

This section provides API allowing to Encrypt and Decrypt Shared key

This section contains the following APIs:

- *HAL_CRYPEx_EncryptSharedKey()*
- *HAL_CRYPEx_DecryptSharedKey()*

15.1.4 Detailed description of functions

HAL_CRYPEx_AESGCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESGCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, const uint32_t * pAuthTag, uint32_t Timeout)

Function description

generate the GCM authentication TAG.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pAuthTag:** Pointer to the authentication buffer the pAuthTag generated here is 128bits length, if the TAG length is less than 128bits, user should consider only the valid part of pAuthTag buffer which correspond exactly to TAG length.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_AESCCM_GenerateAuthTAG

Function name

HAL_StatusTypeDef HAL_CRYPEx_AESCCM_GenerateAuthTAG (CRYP_HandleTypeDef * hcryp, const uint32_t * pAuthTag, uint32_t Timeout)

Function description

AES CCM Authentication TAG generation.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pAuthTag:** Pointer to the authentication buffer the pAuthTag generated here is 128bits length, if the TAG length is less than 128bits, user should consider only the valid part of pAuthTag buffer which correspond exactly to TAG length.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_CRYPEx_UnwrapKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_UnwrapKey (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint32_t Timeout)

Function description

Unwrap (Decrypt) application keys.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the Key buffer to decrypt.
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYPEx_WrapKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_WrapKey (CRYP_HandleTypeDef * hcryp, uint32_t * pInput, uint32_t * pOutput, uint32_t Timeout)

Function description

Wrap (encrypt) application keys.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pInput:** Pointer to the Key buffer to encrypt in case of ECB or CBC
- **pOutput:** Pointer to the Key buffer encrypted in case of ECB or CBC
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYPEx_EncryptSharedKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_EncryptSharedKey (CRYP_HandleTypeDef * hcryp, uint32_t * pKey, uint32_t * pOutput, uint32_t ID, uint32_t Timeout)

Function description

Encrypt Shared key.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pKey:** Pointer to the Key buffer to share
- **pOutput:** Pointer to the Key buffer encrypted
- **ID:** Key share identification
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_CRYPEx_DecryptSharedKey

Function name

HAL_StatusTypeDef HAL_CRYPEx_DecryptSharedKey (CRYP_HandleTypeDef * hcryp, uint32_t * pKey, uint32_t ID, uint32_t Timeout)

Function description

Decrypt Shared key.

Parameters

- **hcryp:** pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
- **pKey:** Pointer to the Key buffer to share
- **ID:** Key share identification
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

16 HAL DAC Generic Driver

16.1 DAC Firmware driver registers structures

16.1.1 __DAC_HandleTypeDef

__DAC_HandleTypeDef is defined in the stm32u5xx_hal_dac.h

Data Fields

- ***DAC_TypeDef * Instance***
- ***__IO HAL_DAC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***DMA_HandleTypeDef * DMA_Handle1***
- ***DMA_HandleTypeDef * DMA_Handle2***
- ***__IO uint32_t ErrorCode***
- ***void(* ConvCpltCallbackCh1***
- ***void(* ConvHalfCpltCallbackCh1***
- ***void(* ErrorCallbackCh1***
- ***void(* DMAUnderrunCallbackCh1***
- ***void(* ConvCpltCallbackCh2***
- ***void(* ConvHalfCpltCallbackCh2***
- ***void(* ErrorCallbackCh2***
- ***void(* DMAUnderrunCallbackCh2***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DAC_TypeDef* __DAC_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_DAC_StateTypeDef __DAC_HandleTypeDef::State***
  DAC communication state
- ***HAL_LockTypeDef __DAC_HandleTypeDef::Lock***
  DAC locking object
- ***DMA_HandleTypeDef* __DAC_HandleTypeDef::DMA_Handle1***
  Pointer DMA handler for channel 1
- ***DMA_HandleTypeDef* __DAC_HandleTypeDef::DMA_Handle2***
  Pointer DMA handler for channel 2
- ***__IO uint32_t __DAC_HandleTypeDef::ErrorCode***
  DAC Error code
- ***void(* __DAC_HandleTypeDef::ConvCpltCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvHalfCpltCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ErrorCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::DMAUnderrunCallbackCh1)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvCpltCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ConvHalfCpltCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::ErrorCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::DMAUnderrunCallbackCh2)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::MspInitCallback)(struct __DAC_HandleTypeDef *hdac)***
- ***void(* __DAC_HandleTypeDef::MspDeInitCallback)(struct __DAC_HandleTypeDef *hdac)***

16.1.2 DAC_SampleAndHoldConfTypeDef

DAC_SampleAndHoldConfTypeDef is defined in the stm32u5xx_hal_dac.h

Data Fields

- ***uint32_t DAC_SampleTime***
- ***uint32_t DAC_HoldTime***
- ***uint32_t DAC_RefreshTime***

Field Documentation

- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_SampleTime***
  Specifies the Sample time for the selected channel. This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_HoldTime***
  Specifies the hold time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 1023
- ***uint32_t DAC_SampleAndHoldConfTypeDef::DAC_RefreshTime***
  Specifies the refresh time for the selected channel This parameter applies when DAC_SampleAndHold is DAC_SAMPLEANDHOLD_ENABLE. This parameter must be a number between Min_Data = 0 and Max_Data = 255

16.1.3 DAC_ChannelConfTypeDef

DAC_ChannelConfTypeDef is defined in the stm32u5xx_hal_dac.h

Data Fields

- ***uint32_t DAC_HighFrequency***
- ***uint32_t DAC_AutonomousMode***
- ***FunctionalState DAC_DMADoubleDataMode***
- ***FunctionalState DAC_SignedFormat***
- ***uint32_t DAC_SampleAndHold***
- ***uint32_t DAC_Trigger***
- ***uint32_t DAC_OutputBuffer***
- ***uint32_t DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_UserTrimming***
- ***uint32_t DAC_TrimmingValue***
- ***DAC_SampleAndHoldConfTypeDef DAC_SampleAndHoldConfig***

Field Documentation

- ***uint32_t DAC_ChannelConfTypeDef::DAC_HighFrequency***
  Specifies the frequency interface mode This parameter can be a value of ***DAC_HighFrequency***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_AutonomousMode***
  Specifies whether the autonomous mode state This parameter can be a value of ***DAC_AutonomousMode***
  Note: **HAL_DACEx_SetConfigAutonomousMode()** API allows to select and update the autonomous mode state afterwards
- ***FunctionalState DAC_ChannelConfTypeDef::DAC_DMADoubleDataMode***
  Specifies if DMA double data mode should be enabled or not for the selected channel. This parameter can be ENABLE or DISABLE
- ***FunctionalState DAC_ChannelConfTypeDef::DAC_SignedFormat***
  Specifies if signed format should be used or not for the selected channel. This parameter can be ENABLE or DISABLE
- ***uint32_t DAC_ChannelConfTypeDef::DAC_SampleAndHold***
  Specifies whether the DAC mode. This parameter can be a value of ***DAC_SampleAndHold***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_Trigger***
  Specifies the external trigger for the selected DAC channel. This parameter can be a value of ***DAC_trigger_selection***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer***
  Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of ***DAC_output_buffer***

- ***uint32_t DAC_ChannelConfTypeDef::DAC_ConnectOnChipPeripheral***
  Specifies whether the DAC output is connected or not to on chip peripheral. This parameter can be a value of ***DAC_ConnectOnChipPeripheral***
- ***uint32_t DAC_ChannelConfTypeDef::DAC_UserTrimming***
  Specifies the trimming mode This parameter must be a value of ***DAC_UserTrimming*** DAC_UserTrimming is either factory or user trimming
- ***uint32_t DAC_ChannelConfTypeDef::DAC_TrimmingValue***
  Specifies the offset trimming value i.e. when DAC_SampleAndHold is DAC_TRIMMING_USER. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***DAC_SampleAndHoldConfTypeDef DAC_ChannelConfTypeDef::DAC_SampleAndHoldConfig***
  Sample and Hold settings

16.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

16.2.1 DAC Peripheral features

DAC Channels

STM32U5 devices integrate two 12-bit Digital Analog Converters The 2 converters (i.e. channel1 & channel2) can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output or connected to on-chip peripherals (ex. ADC).
2. DAC channel2 with DAC_OUT2 (PA5) as output or connected to on-chip peripherals (ex. ADC).

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9. The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM1, TIM2, TIM4, TIM5, TIM6, TIM7, TIM8 and TIM15 (DAC_TRIGGER_T1_TRGO, DAC_TRIGGER_T2_TRGO...)
3. Low Power Timers TRGO: LPTIM1 and LPTIM3 (DAC_TRIGGER_LPTIM1_OUT, DAC_TRIGGER_LPTIM3_OUT)
4. Software using DAC_TRIGGER_SOFTWARE

The trigger selection depends on the PWR mode: in stop0, stop1 and stop2 we should select DAC_TRIGGER_EXT_IT9, DAC_TRIGGER_LPTIM1_OUT or DAC_TRIGGER_LPTIM3_OUT.The other triggers are not functional.

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;

Note: *Refer to the device datasheet for more details about output impedance value with and without output buffer.*

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel2 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC Sample and Hold feature

For each converter, 2 modes are supported: normal mode and "sample and hold" mode (i.e. low power mode). In the sample and hold mode, the DAC core converts data, then holds the converted voltage on a capacitor. When not converting, the DAC cores and buffer are completely turned off between samples and the DAC output is tri-stated, therefore reducing the overall power consumption. A new stabilization period is needed before each new conversion. The sample and hold allow setting internal or external voltage @ low power consumption cost (output value can be at any given rate either by CPU or DMA). The Sample and hold block and registers uses either LSI & run in several power modes: run mode, sleep mode, low power run, low power sleep mode & stop1 mode. Low power stop1 mode allows only static conversion. To enable Sample and Hold mode Enable LSI using HAL_RCC_OscConfig with RCC_OSCILLATORTYPE_LSI & RCC_LSI_ON parameters. Use DAC_InitStructure.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_ENABLE; & DAC_ChannelConfTypeDef.DAC_SampleAndHoldConfig.DAC_SampleTime, DAC_HoldTime & DAC_RefreshTime;

DAC calibration feature

1. The 2 converters (channel1 & channel2) provide calibration capabilities.
   – Calibration aims at correcting some offset of output buffer.
   – The DAC uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   – The user defined settings can be figured out using self calibration handled by HAL_DACEx_SelfCalibrate.
   – HAL_DACEx_SelfCalibrate:
     ◦ Runs automatically the calibration.
     ◦ Enables the user trimming mode
     ◦ Updates a structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave
2. Triangle wave

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondence

The analog output voltage on each DAC channel pin is determined by the following equation:

DAC_OUTx = VREF+ * DOR / 4095

- with DOR is the Data Output Register

VREF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, DAC_OUT1 = (3.3 * 868) / 4095 = 0.7V

DMA requests

A DMA request can be generated when an external trigger (but not a software trigger) occurs if DMA requests are enabled using HAL_DAC_Start_DMA(). DMA requests are mapped as following: GPDMA requests are mapped as following:

- DAC channel1 mapped on GPDMA request 2 (can be any GPDMA channel)

- DAC channel2 mapped on GPDMA request 3 (can be any GPDMA channel) LPDMA requests are mapped as following:
- DAC channel1 mapped on LPDMA request 8 (can be any LPDMA channel)
- DAC channel2 mapped on LPDMA request 9 (can be any LPDMA channel)

High frequency interface mode

The high frequency interface informs DAC instance about the bus frequency in use. It is mandatory information for DAC (as internal timing of DAC is bus frequency dependent) provided thanks to parameter DAC_HighFrequency handled in HAL_DAC_ConfigChannel () function. Use of DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC value of DAC_HighFrequency is recommended function figured out the correct setting. The high frequency mode is same for all converters of a same DAC instance. Either same parameter DAC_HighFrequency is used for all DAC converters or again self DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC detection parameter.

Note: For Dual mode and specific signal (Triangle and noise) generation please refer to Extended Features Driver description

16.2.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA() functions.

Calibration mode IO operation

- Retrieve the factory trimming (calibration settings) using HAL_DACEx_GetTrimOffset()
- Run the calibration using HAL_DACEx_SelfCalibrate()
- Update the trimming while DAC running using HAL_DACEx_SetUserTrimming()

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion First issued trigger will start the conversion of the value previously set by HAL_DAC_SetValue().
- At the middle of data transfer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2()
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() and add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1()
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

Callback registration

The compilation define USE_HAL_DAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_DAC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_DAC_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- ConvCpltCallbackCh1 : callback when a half transfer is completed on Ch1.
- ConvHalfCpltCallbackCh1 : callback when a transfer is completed on Ch1.
- ErrorCallbackCh1 : callback when an error occurs on Ch1.
- DMAUnderrunCallbackCh1 : callback when an underrun error occurs on Ch1.
- ConvCpltCallbackCh2 : callback when a half transfer is completed on Ch2.
- ConvHalfCpltCallbackCh2 : callback when a transfer is completed on Ch2.
- ErrorCallbackCh2 : callback when an error occurs on Ch2.
- DMAUnderrunCallbackCh2 : callback when an underrun error occurs on Ch2.
- MspInitCallback : DAC MspInit.
- MspDeInitCallback : DAC MspdeInit.
- All Callbacks This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_DAC_Init and if the state is HAL_DAC_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_DAC_Init and HAL_DAC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_DAC_Init and HAL_DAC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DAC_RegisterCallback before calling HAL_DAC_DeInit or HAL_DAC_Init function. When The compilation define USE_HAL_DAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

DAC HAL driver macros list

Below the list of most used macros in DAC HAL driver.

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status

Note: *You can refer to the DAC HAL driver header file for more useful macros*

16.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:
- *HAL_DAC_Init()*
- *HAL_DAC_DeInit()*
- *HAL_DAC_MspInit()*
- *HAL_DAC_MspDeInit()*
- *HAL_DAC_RegisterCallback()*
- *HAL_DAC_UnRegisterCallback()*

16.2.4 IO operation functions

This section provides functions allowing to:
- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.

This section contains the following APIs:
- *HAL_DAC_Start()*
- *HAL_DAC_Stop()*
- *HAL_DAC_Start_DMA()*
- *HAL_DAC_Stop_DMA()*
- *HAL_DAC_IRQHandler()*
- *HAL_DAC_SetValue()*
- *HAL_DAC_ConvCpltCallbackCh1()*
- *HAL_DAC_ConvHalfCpltCallbackCh1()*
- *HAL_DAC_ErrorCallbackCh1()*
- *HAL_DAC_DMAUnderrunCallbackCh1()*
- *HAL_DAC_RegisterCallback()*
- *HAL_DAC_UnRegisterCallback()*

16.2.5 Peripheral Control functions

This section provides functions allowing to:
- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:
- *HAL_DAC_GetValue()*
- *HAL_DAC_ConfigChannel()*

16.2.6 Peripheral State and Errors functions

This subsection provides functions allowing to
- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:
- *HAL_DAC_GetState()*
- *HAL_DAC_GetError()*

16.2.7 Detailed description of functions

HAL_DAC_Init

Function name

HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC peripheral according to the specified parameters in the DAC_InitStruct and initialize the associated handle.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_DeInit

Function name

HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)

Function description

Deinitialize the DAC peripheral registers to their default reset values.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DAC_MspInit

Function name

void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)

Function description

Initialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_MspDeInit

Function name

void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)

Function description

DeInitialize the DAC MSP.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_Start

Function name

HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Stop

Function name

HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_Start_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, const uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DAC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion of channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DAC_IRQHandler

Function name

void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)

Function description

Handles DAC interrupt request This function uses the interruption of DMA underrun.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_SetValue

Function name

HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

Function description

Set the specified data holding register value for DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **Alignment:** Specifies the data alignment. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data:** Data to be loaded in the selected data holding register.

Return values

- **HAL:** status

HAL_DAC_ConvCpltCallbackCh1

Function name

void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ConvHalfCpltCallbackCh1

Function name

void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_ErrorCallbackCh1

Function name

void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_DMAUnderrunCallbackCh1

Function name

void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for channel1.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DAC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DAC_RegisterCallback (DAC_HandleTypeDef * hdac, HAL_DAC_CallbackIDTypeDef CallbackID, pDAC_CallbackTypeDef pCallback)

Function description

Register a User DAC Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hdac:** DAC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_DAC_ERROR_INVALID_CALLBACK DAC Error Callback ID
  – HAL_DAC_CH1_COMPLETE_CB_ID DAC CH1 Complete Callback ID
  – HAL_DAC_CH1_HALF_COMPLETE_CB_ID DAC CH1 Half Complete Callback ID
  – HAL_DAC_CH1_ERROR_ID DAC CH1 Error Callback ID
  – HAL_DAC_CH1_UNDERRUN_CB_ID DAC CH1 UnderRun Callback ID
  – HAL_DAC_CH2_COMPLETE_CB_ID DAC CH2 Complete Callback ID
  – HAL_DAC_CH2_HALF_COMPLETE_CB_ID DAC CH2 Half Complete Callback ID
  – HAL_DAC_CH2_ERROR_ID DAC CH2 Error Callback ID
  – HAL_DAC_CH2_UNDERRUN_CB_ID DAC CH2 UnderRun Callback ID
  – HAL_DAC_MSPINIT_CB_ID DAC MSP Init Callback ID
  – HAL_DAC_MSPDEINIT_CB_ID DAC MSP DeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_DAC_RegisterCallback() may be called before HAL_DAC_Init() in HAL_DAC_STATE_RESET to register callbacks for HAL_DAC_MSPINIT_CB_ID and HAL_DAC_MSPDEINIT_CB_ID

HAL_DAC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DAC_UnRegisterCallback (DAC_HandleTypeDef * hdac, HAL_DAC_CallbackIDTypeDef CallbackID)

Function description

Unregister a User DAC Callback DAC Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hdac:** DAC handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_DAC_CH1_COMPLETE_CB_ID DAC CH1 transfer Complete Callback ID
  - HAL_DAC_CH1_HALF_COMPLETE_CB_ID DAC CH1 Half Complete Callback ID
  - HAL_DAC_CH1_ERROR_ID DAC CH1 Error Callback ID
  - HAL_DAC_CH1_UNDERRUN_CB_ID DAC CH1 UnderRun Callback ID
  - HAL_DAC_CH2_COMPLETE_CB_ID DAC CH2 Complete Callback ID
  - HAL_DAC_CH2_HALF_COMPLETE_CB_ID DAC CH2 Half Complete Callback ID
  - HAL_DAC_CH2_ERROR_ID DAC CH2 Error Callback ID
  - HAL_DAC_CH2_UNDERRUN_CB_ID DAC CH2 UnderRun Callback ID
  - HAL_DAC_MSPINIT_CB_ID DAC MSP Init Callback ID
  - HAL_DAC_MSPDEINIT_CB_ID DAC MSP DeInit Callback ID
  - HAL_DAC_ALL_CB_ID DAC All callbacks

Return values

- **status:**

Notes

- The HAL_DAC_UnRegisterCallback() may be called before HAL_DAC_Init() in HAL_DAC_STATE_RESET to un-register callbacks for HAL_DAC_MSPINIT_CB_ID and HAL_DAC_MSPDEINIT_CB_ID

HAL_DAC_GetValue

Function name

uint32_t HAL_DAC_GetValue (const DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Returns the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **The:** selected DAC channel data output value.

HAL_DAC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, const DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Configures the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  – DAC_CHANNEL_1: DAC Channel1 selected
  – DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

Notes

- By calling this function, if the application will be run in stop mode, the DAC_Trigger field of the structure sConfig should be one of the following values: DAC_TRIGGER_STOP_LPTIM1_OUTDAC_TRIGGER_STOP_LPTIM3_OUTDAC_TRIGGER_STOP_EXT_IT9 For application where DAC is not run in stop mode, the DAC trigger should be a value of DAC trigger selection
- By calling this function, the high frequency interface mode (HFSEL bits) will be set. This parameter scope is the DAC instance. As the function is called for each channel, the DAC high frequency interface mode of sConfig must be the same at each call. (or DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC self detect).

HAL_DAC_GetState

Function name

HAL_DAC_StateTypeDef HAL_DAC_GetState (const DAC_HandleTypeDef * hdac)

Function description

return the DAC handle state

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** state

HAL_DAC_GetError

Function name

uint32_t HAL_DAC_GetError (const DAC_HandleTypeDef * hdac)

Function description

Return the DAC error code.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **DAC:** Error Code

DAC_DMAConvCpltCh1

Function name

void DAC_DMAConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh1

Function name

void DAC_DMAErrorCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh1

Function name

void DAC_DMAHalfConvCpltCh1 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

16.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

16.3.1 DAC

DAC

DAC Autonomous Mode

DAC_AUTONOMOUS_MODE_DISABLE

Autonomous mode disable

DAC_AUTONOMOUS_MODE_ENABLE

Autonomous mode enable

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC ConnectOnChipPeripheral

DAC_CHIPCONNECT_EXTERNAL

DAC_CHIPCONNECT_INTERNAL

DAC_CHIPCONNECT_BOTH

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE

No error

HAL_DAC_ERROR_DMAUNDERRUNCH1

DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERRUNCH2

DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA

DMA error

HAL_DAC_ERROR_TIMEOUT

Timeout error

HAL_DAC_ERROR_INVALID_CALLBACK

Invalid callback error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC channel.

Return value:

- None

DAC_DHR12R1_ALIGNMENT

Description:

- Set DHR12R1 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12R2_ALIGNMENT

Description:

- Set DHR12R2 alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

DAC_DHR12RD_ALIGNMENT

Description:

- Set DHR12RD alignment.

Parameters:

- __ALIGNMENT__: specifies the DAC alignment

Return value:

- None

__HAL_DAC_ENABLE_IT

Description:

- Enable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  – DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  – DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_DISABLE_IT

Description:

- Disable the DAC interrupt.

Parameters:

- __HANDLE__: specifies the DAC handle
- __INTERRUPT__: specifies the DAC interrupt. This parameter can be any combination of the following values:
  – DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  – DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- None

__HAL_DAC_GET_IT_SOURCE

Description:

- Check whether the specified DAC interrupt source is enabled or not.

Parameters:

- __HANDLE__: DAC handle
- __INTERRUPT__: DAC interrupt source to check This parameter can be any combination of the following values:
  – DAC_IT_DMAUDR1 DAC channel 1 DMA underrun interrupt
  – DAC_IT_DMAUDR2 DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

__HAL_DAC_GET_FLAG

Description:

- Get the selected DAC's flag status.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to get. This parameter can be any combination of the following values:
  – DAC_FLAG_DMAUDR1 DAC channel 1 DMA underrun flag
  – DAC_FLAG_DMAUDR2 DAC channel 2 DMA underrun flag
  – DAC_FLAG_DAC1RDY DAC channel 1 ready status flag
  – DAC_FLAG_DAC2RDY DAC channel 2 ready status flag

Return value:

- None

__HAL_DAC_CLEAR_FLAG

Description:

- Clear the DAC's flag.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __FLAG__: specifies the DAC flag to clear. This parameter can be any combination of the following values:
  – DAC_FLAG_DMAUDR1 DAC channel 1 DMA underrun flag
  – DAC_FLAG_DMAUDR2 DAC channel 2 DMA underrun flag

Return value:

- None

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

DAC_FLAG_DAC1RDY

DAC_FLAG_DAC2RDY

DAC high frequency interface mode

DAC_HIGH_FREQUENCY_INTERFACE_MODE_DISABLE

High frequency interface mode disabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_80MHZ

High frequency interface mode compatible to AHB>80MHz enabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_ABOVE_160MHZ

High frequency interface mode compatible to AHB>160MHz enabled

DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC

High frequency interface mode automatic

DAC IT definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC power mode

DAC_SAMPLEANDHOLD_DISABLE

DAC_SAMPLEANDHOLD_ENABLE

DAC trigger selection

DAC_TRIGGER_NONE

conversion is automatic once the DAC_DHRxxxx register has been loaded, and not by external trigger

DAC_TRIGGER_SOFTWARE

conversion started by software trigger for DAC channel

DAC_TRIGGER_T1_TRGO

TIM1 TRGO selected as external conversion trigger for DAC channel.

DAC_TRIGGER_T2_TRGO

TIM2 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T4_TRGO

TIM4 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T5_TRGO

TIM5 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T6_TRGO

TIM6 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T7_TRGO

TIM7 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T8_TRGO

TIM8 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_T15_TRGO

TIM15 TRGO selected as external conversion trigger for DAC channel

DAC_TRIGGER_LPTIM1_CH1

LPTIM1 CH1 selected as external conversion trigger for DAC channel

DAC_TRIGGER_LPTIM3_CH1

LPTIM3 CH1 selected as external conversion trigger for DAC channel

DAC_TRIGGER_EXT_IT9

EXTI Line9 event selected as external conversion trigger for DAC channel

DAC Trigger Stop Mode

DAC_TRIGGER_STOP_LPTIM1_CH1

LPTIM1 output selected as DAC trigger in stop mode

DAC_TRIGGER_STOP_LPTIM3_CH1

LPTIM3 output selected as DAC trigger in stop mode

DAC_TRIGGER_STOP_EXT_IT9

EXTI line 9 selected as DAC trigger in stop mode

DAC User Trimming

DAC_TRIMMING_FACTORY

Factory trimming

DAC_TRIMMING_USER

User trimming

17 HAL DAC Extension Driver

17.1 DACEx Firmware driver registers structures

17.1.1 DAC_AutonomousModeConfTypeDef

DAC_AutonomousModeConfTypeDef is defined in the stm32u5xx_hal_dac_ex.h

Data Fields

- ***uint32_t AutonomousModeState***

Field Documentation

- ***uint32_t DAC_AutonomousModeConfTypeDef::AutonomousModeState***
  Specifies the autonomous mode state. This parameter can be a value of ***DAC_AutonomousMode***

17.2 DACEx Firmware driver API description

The following section lists the various functions of the DACEx library.

17.2.1 How to use this driver

Dual mode IO operation

- Use HAL_DACEx_DualStart() to enable both channel and start conversion for dual mode operation. If software trigger is selected, using HAL_DACEx_DualStart() will start the conversion of the value previously set by HAL_DACEx_DualSetValue().
- Use HAL_DACEx_DualStop() to disable both channel and stop conversion for dual mode operation.
- Use HAL_DACEx_DualStart_DMA() to enable both channel and start conversion for dual mode operation using DMA to feed DAC converters. First issued trigger will start the conversion of the value previously set by HAL_DACEx_DualSetValue(). The same callbacks that are used in single mode are called in dual mode to notify transfer completion (half complete or complete), errors or underrun.
- Use HAL_DACEx_DualStop_DMA() to disable both channel and stop conversion for dual mode operation using DMA to feed DAC converters.
- When Dual mode is enabled (i.e. DAC Channel1 and Channel2 are used simultaneously) : Use HAL_DACEx_DualGetValue() to get digital data to be converted and use HAL_DACEx_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.

Signal generation operation

- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.
- HAL_DACEx_SelfCalibrate to calibrate one DAC channel.
- HAL_DACEx_SetUserTrimming to set user trimming value.
- HAL_DACEx_GetTrimOffset to retrieve trimming value (factory setting after reset, user setting if HAL_DACEx_SetUserTrimming have been used at least one time after reset).

Autonomous Mode operation

- Use HAL_DACx_SetConfigAutonomousMode() to configure the autonomous mode
- Use HAL_DACx_GetConfigAutonomousMode() to get the current configuration of the autonomous mode
- Use HAL_DACx_ClearConfigAutonomousMode() to clear the configuration of the autonomous mode

17.2.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.

- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- *HAL_DACEx_DualStart()*
- *HAL_DACEx_DualStop()*
- *HAL_DACEx_DualStart_DMA()*
- *HAL_DACEx_DualStop_DMA()*
- *HAL_DACEx_TriangleWaveGenerate()*
- *HAL_DACEx_NoiseWaveGenerate()*
- *HAL_DACEx_DualSetValue()*
- *HAL_DACEx_ConvCpltCallbackCh2()*
- *HAL_DACEx_ConvHalfCpltCallbackCh2()*
- *HAL_DACEx_ErrorCallbackCh2()*
- *HAL_DACEx_DMAUnderrunCallbackCh2()*
- *HAL_DACEx_SelfCalibrate()*
- *HAL_DACEx_SetUserTrimming()*
- *HAL_DACEx_GetTrimOffset()*
- *HAL_DACEx_DualGetValue()*

17.2.3 Peripheral Control functions

This section provides functions allowing to:

- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- *HAL_DACEx_DualGetValue()*
- *HAL_DACEx_SelfCalibrate()*
- *HAL_DACEx_SetUserTrimming()*
- *HAL_DACEx_GetTrimOffset()*
- *HAL_DACEx_SetConfigAutonomousMode()*
- *HAL_DACEx_GetConfigAutonomousMode()*
- *HAL_DACEx_ClearConfigAutonomousMode()*

17.2.4 Detailed description of functions

HAL_DACEx_TriangleWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  – DAC_CHANNEL_1: DAC Channel1 selected
  – DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Select max triangle amplitude. This parameter can be one of the following values:
  – DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1
  – DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3
  – DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7
  – DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15
  – DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31
  – DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63
  – DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127
  – DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255
  – DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511
  – DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023
  – DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047
  – DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095

Return values

- **HAL:** status

HAL_DACEx_NoiseWaveGenerate

Function name

HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)

Function description

Enable or disable the selected DAC channel wave generation.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  – DAC_CHANNEL_1: DAC Channel1 selected
  – DAC_CHANNEL_2: DAC Channel2 selected
- **Amplitude:** Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values:
  – DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation
  – DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS5_0: Unmask DAC channel LFSR bit[5:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
  – DAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values

- **HAL:** status

HAL_DACEx_DualStart

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart (DAC_HandleTypeDef * hdac)

Function description

Enables DAC and starts conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStop

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop (DAC_HandleTypeDef * hdac)

Function description

Disables DAC and stop conversion of both channels.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

HAL_DACEx_DualStart_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStart_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, const uint32_t * pData, uint32_t Length, uint32_t Alignment)

Function description

Enables DAC and starts conversion of both channel 1 and 2 of the same DAC.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that will request data from DMA. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **pData:** The destination peripheral Buffer address.
- **Length:** The length of data to be transferred from memory to DAC peripheral
- **Alignment:** Specifies the data alignment for DAC channel. This parameter can be one of the following values:
  - DAC_ALIGN_8B_R: 8bit right data alignment selected
  - DAC_ALIGN_12B_L: 12bit left data alignment selected
  - DAC_ALIGN_12B_R: 12bit right data alignment selected

Return values

- **HAL:** status

HAL_DACEx_DualStop_DMA

Function name

HAL_StatusTypeDef HAL_DACEx_DualStop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Disables DAC and stop conversion both channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Channel:** The DAC channel that requests data from DMA. This parameter can be one of the following values:
  – DAC_CHANNEL_1: DAC Channel1 selected
  – DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **HAL:** status

HAL_DACEx_DualSetValue

Function name

HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)

Function description

Set the specified data holding register value for dual DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selected DAC_ALIGN_12B_L: 12bit left data alignment selected DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel1 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel2 to be loaded in the selected data holding register.

Return values

- **HAL:** status

Notes

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

HAL_DACEx_DualGetValue

Function name

uint32_t HAL_DACEx_DualGetValue (const DAC_HandleTypeDef * hdac)

Function description

Return the last data output value of the selected DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **The:** selected DAC channel data output value.

HAL_DACEx_ConvCpltCallbackCh2

Function name

void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion complete callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ConvHalfCpltCallbackCh2

Function name

void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Conversion half DMA transfer callback in non-blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_ErrorCallbackCh2

Function name

void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

Error DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_DMAUnderrunCallbackCh2

Function name

void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef * hdac)

Function description

DMA underrun DAC callback for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **None:**

HAL_DACEx_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_DACEx_SelfCalibrate (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)

Function description

Run the self calibration of one DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC channel configuration structure.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Updates:** DAC_TrimmingValue. , DAC_UserTrimming set to DAC_UserTrimming
- **HAL:** status

Notes

- Calibration runs about 7 ms.

HAL_DACEx_SetUserTrimming

Function name

HAL_StatusTypeDef HAL_DACEx_SetUserTrimming (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel, uint32_t NewTrimmingValue)

Function description

Set the trimming mode and trimming value (user trimming mode applied).

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** DAC configuration structure updated with new DAC trimming value.
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  - DAC_CHANNEL_1: DAC Channel1 selected
  - DAC_CHANNEL_2: DAC Channel2 selected
- **NewTrimmingValue:** DAC new trimming value

Return values

- **HAL:** status

HAL_DACEx_GetTrimOffset

Function name

uint32_t HAL_DACEx_GetTrimOffset (const DAC_HandleTypeDef * hdac, uint32_t Channel)

Function description

Return the DAC trimming value.

Parameters

- **hdac:** DAC handle
- **Channel:** The selected DAC channel. This parameter can be one of the following values:
  – DAC_CHANNEL_1: DAC Channel1 selected
  – DAC_CHANNEL_2: DAC Channel2 selected

Return values

- **Trimming:** value : range: 0->31

HAL_DACEx_SetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_DACEx_SetConfigAutonomousMode (DAC_HandleTypeDef * hdac, const DAC_AutonomousModeConfTypeDef * sConfig)

Function description

Set autonomous mode Configuration.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** pointer to Autonomous mode structure parameters.

Return values

- **HAL:** status

Notes

- The autonomous mode applies to the 2 channels of a DAC block (same for both channels)

HAL_DACEx_GetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_DACEx_GetConfigAutonomousMode (const DAC_HandleTypeDef * hdac, DAC_AutonomousModeConfTypeDef * sConfig)

Function description

Get autonomous mode Configuration.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **sConfig:** pointer to Autonomous mode structure parameters.

Return values

- **HAL:** status

HAL_DACEx_ClearConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_DACEx_ClearConfigAutonomousMode (DAC_HandleTypeDef * hdac)

Function description

Clear autonomous mode Configuration.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- **HAL:** status

Notes

- The autonomous mode applies to the 2 channels of a DAC block (same for both channels)

DAC_DMAConvCpltCh2

Function name

void DAC_DMAConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA conversion complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAErrorCh2

Function name

void DAC_DMAErrorCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA error callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

DAC_DMAHalfConvCpltCh2

Function name

void DAC_DMAHalfConvCpltCh2 (DMA_HandleTypeDef * hdma)

Function description

DMA half transfer complete callback.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.

Return values

- **None:**

17.3 DACEx Firmware driver defines

The following section lists the various define and macros of the module.

17.3.1 DACEx

DACEx

DACEx lfsrunmask triangle amplitude

DAC_LFSRUNMASK_BIT0

Unmask DAC channel LFSR bit0 for noise wave generation

DAC_LFSRUNMASK_BITS1_0

Unmask DAC channel LFSR bit[1:0] for noise wave generation

DAC_LFSRUNMASK_BITS2_0

Unmask DAC channel LFSR bit[2:0] for noise wave generation

DAC_LFSRUNMASK_BITS3_0

Unmask DAC channel LFSR bit[3:0] for noise wave generation

DAC_LFSRUNMASK_BITS4_0

Unmask DAC channel LFSR bit[4:0] for noise wave generation

DAC_LFSRUNMASK_BITS5_0

Unmask DAC channel LFSR bit[5:0] for noise wave generation

DAC_LFSRUNMASK_BITS6_0

Unmask DAC channel LFSR bit[6:0] for noise wave generation

DAC_LFSRUNMASK_BITS7_0

Unmask DAC channel LFSR bit[7:0] for noise wave generation

DAC_LFSRUNMASK_BITS8_0

Unmask DAC channel LFSR bit[8:0] for noise wave generation

DAC_LFSRUNMASK_BITS9_0

Unmask DAC channel LFSR bit[9:0] for noise wave generation

DAC_LFSRUNMASK_BITS10_0

Unmask DAC channel LFSR bit[10:0] for noise wave generation

DAC_LFSRUNMASK_BITS11_0

Unmask DAC channel LFSR bit[11:0] for noise wave generation

DAC_TRIANGLEAMPLITUDE_1

Select max triangle amplitude of 1

DAC_TRIANGLEAMPLITUDE_3

Select max triangle amplitude of 3

DAC_TRIANGLEAMPLITUDE_7

Select max triangle amplitude of 7

DAC_TRIANGLEAMPLITUDE_15

Select max triangle amplitude of 15

DAC_TRIANGLEAMPLITUDE_31

Select max triangle amplitude of 31

DAC_TRIANGLEAMPLITUDE_63

Select max triangle amplitude of 63

DAC_TRIANGLEAMPLITUDE_127

Select max triangle amplitude of 127

DAC_TRIANGLEAMPLITUDE_255

Select max triangle amplitude of 255

DAC_TRIANGLEAMPLITUDE_511

Select max triangle amplitude of 511

DAC_TRIANGLEAMPLITUDE_1023

Select max triangle amplitude of 1023

DAC_TRIANGLEAMPLITUDE_2047

Select max triangle amplitude of 2047

DAC_TRIANGLEAMPLITUDE_4095

Select max triangle amplitude of 4095

18 HAL DCACHE Generic Driver

18.1 DCACHE Firmware driver registers structures

18.1.1 DCACHE_InitTypeDef

DCACHE_InitTypeDef is defined in the stm32u5xx_hal_dcache.h

Data Fields

- ***uint32_t ReadBurstType***

Field Documentation

- ***uint32_t DCACHE_InitTypeDef::ReadBurstType***
  Burst type to be applied for Data Cache This parameter can be a value of ***DCACHE_Read_Burst_Type***

18.1.2 __DCACHE_HandleTypeDef

__DCACHE_HandleTypeDef is defined in the stm32u5xx_hal_dcache.h

Data Fields

- ***DCACHE_TypeDef * Instance***
- ***DCACHE_InitTypeDef Init***
- ***void(* ErrorCallback***
- ***void(* CleanByAddrCallback***
- ***void(* InvalidateByAddrCallback***
- ***void(* InvalidateCompleteCallback***
- ***void(* CleanAndInvalidateByAddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***__IO HAL_DCACHE_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DCACHE_TypeDef* __DCACHE_HandleTypeDef::Instance***
  DCACHE register base address.
- ***DCACHE_InitTypeDef __DCACHE_HandleTypeDef::Init***
  DCACHE Initialization Structure.
- ***void(* __DCACHE_HandleTypeDef::ErrorCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::CleanByAddrCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::InvalidateByAddrCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::InvalidateCompleteCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::CleanAndInvalidateByAddrCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::MspInitCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***void(* __DCACHE_HandleTypeDef::MspDeInitCallback)(struct __DCACHE_HandleTypeDef *hdcache)***
- ***__IO HAL_DCACHE_StateTypeDef __DCACHE_HandleTypeDef::State***
- ***__IO uint32_t __DCACHE_HandleTypeDef::ErrorCode***

18.2 DCACHE Firmware driver API description

The following section lists the various functions of the DCACHE library.

18.2.1 How to use this driver

1. Configure and enable the MPU to override default config if needed, please refers to ARM manual for default memory attribute. Then enable DCACHE.

- Use HAL_DCACHE_Invalidate() to invalidate the full cache content:
  - Cache content is lost, and reloaded when needed.
  - Used for complete invalidate of the DCACHE in case.
  - Blocking call until operation is done.
- Use HAL_DCACHE_InvalidateByAddr() to invalidate cache content for specific range:
  - Cache content for specific range is lost, and reloaded when needed.
  - Used when excepting a buffer to be updated by a peripheral (typically DMA transfer)
  - Blocking call until operation is done.
- Use HAL_DCACHE_CleanByAddr() to clean cache content for a specific range:
  - Cache content for specific range is written back to memory.
  - Used when buffer is updated by CPU before usage by a peripheral (typically DMA transfer)
  - Blocking call until operation is done.
- Use HAL_DCACHE_CleanInvalidateByAddr() to clean and invalidate cache content for a specific range:
  - Cache content for specific range is written back to memory, and reloaded when needed.
  - Used when sharing buffer between CPU and other peripheral.
  - Recommended to use for MPU reprogramming.
  - Blocking call until operation is done.

Interrupt mode IO operation

- Configure the DCACHE interrupt priority using HAL_NVIC_SetPriority()
- Enable the DCACHE IRQ handler using HAL_NVIC_EnableIRQ()
- Override weak definition for following callbacks (if needed):
  - HAL_DCACHE_CleanAndInvalidateByAddrCallback()
  - HAL_DCACHE_InvalidateCompleteCallback()
  - HAL_DCACHE_InvalidateByAddrCallback()
  - HAL_DCACHE_CleanByAddrCallback()
  - HAL_DCACHE_ErrorCallback()
- Use HAL_DCACHE_<COMMAND>_IT() to start a DCACHE operation with IT enabled.
- Use HAL_DCACHE_IRQHandler() called under DCACHEx_IRQHandler() Interrupt subroutine

Use HAL_DCACHE_GetState() function to return the DCACHE state and HAL_DCACHE_GetError() in case of error detection.

DCACHE HAL driver macros list

Below the list of macros defined in the DCACHE HAL driver.

- __HAL_DCACHE_ENABLE_IT : Enable DCACHE interrupts.
- __HAL_DCACHE_DISABLE_IT : Disable DCACHE interrupts.
- __HAL_DCACHE_GET_IT_SOURCE: Check whether the specified DCACHE interrupt source is enabled or not.
- __HAL_DCACHE_GET_FLAG : Check whether the selected DCACHE flag is set or not.
- __HAL_DCACHE_CLEAR_FLAG : Clear the selected DCACHE flags.

Note: You can refer to the header file of the DCACHE HAL driver for more useful macros.

18.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the DCACHEx peripheral:

- User must implement HAL_DCACHE_MspInit() function in which he configures all related peripherals resources (CLOCK, MPU, IT and NVIC).

- Call the function HAL_DCACHE_Init() to configure the selected device with the selected configuration:
  – ReadBurstType
- Call the function HAL_DCACHE_DeInit() to restore the reset configuration of the selected DCACHEx peripheral.

This section contains the following APIs:
- *HAL_DCACHE_Init()*
- *HAL_DCACHE_DeInit()*
- *HAL_DCACHE_MspInit()*
- *HAL_DCACHE_MspDeInit()*

18.2.3 IO operation functions

This section provides functions allowing to:
- Enable the Data cache.
- Disable the Data cache.
- Set Read Burst Type.
- Invalidate the Data cache.
- Invalidate the Data cache with interrupt.
- Clean the Data cache by Addr.
- Invalidate the Data cache by Addr.
- Clean and Invalidate the Data cache by Addr.
- Clean the Data cache by Addr with interrupt.
- Invalidate the Data cache by Addr with interrupt.
- Clean and Invalidate the Data cache by Addr with interrupt.
- Start the Data Cache performance monitoring.
- Stop the Data Cache performance monitoring.
- Reset the Data Cache performance monitoring values.
- Get the Data Cache performance Read Hit monitoring value.
- Get the Data Cache performance Read Miss monitoring value.
- Get the Data Cache performance Write Hit monitoring value.
- Get the Data Cache performance Write Miss monitoring value.

This section contains the following APIs:
- *HAL_DCACHE_Enable()*
- *HAL_DCACHE_Disable()*
- *HAL_DCACHE_IsEnabled()*
- *HAL_DCACHE_SetReadBurstType()*
- *HAL_DCACHE_Invalidate()*
- *HAL_DCACHE_InvalidateByAddr()*
- *HAL_DCACHE_CleanByAddr()*
- *HAL_DCACHE_CleanInvalidByAddr()*
- *HAL_DCACHE_Invalidate_IT()*
- *HAL_DCACHE_InvalidateByAddr_IT()*
- *HAL_DCACHE_CleanByAddr_IT()*
- *HAL_DCACHE_CleanInvalidByAddr_IT()*
- *HAL_DCACHE_Monitor_Start()*
- *HAL_DCACHE_Monitor_Stop()*
- *HAL_DCACHE_Monitor_Reset()*
- *HAL_DCACHE_Monitor_GetReadHitValue()*
- *HAL_DCACHE_Monitor_GetReadMissValue()*
- *HAL_DCACHE_Monitor_GetWriteHitValue()*
- *HAL_DCACHE_Monitor_GetWriteMissValue()*

- *HAL_DCACHE_IRQHandler()*
- *HAL_DCACHE_ErrorCallback()*
- *HAL_DCACHE_CleanByAddrCallback()*
- *HAL_DCACHE_InvalidateByAddrCallback()*
- *HAL_DCACHE_InvalidateCompleteCallback()*
- *HAL_DCACHE_CleanAndInvalidateByAddrCallback()*
- *HAL_DCACHE_RegisterCallback()*
- *HAL_DCACHE_UnRegisterCallback()*

18.2.4 DCACHE IRQ Handler and Callback functions

This section provides functions allowing to treat ISR and provide user callback
This section contains the following APIs:

- *HAL_DCACHE_IRQHandler()*
- *HAL_DCACHE_RegisterCallback()*
- *HAL_DCACHE_UnRegisterCallback()*
- *HAL_DCACHE_CleanByAddrCallback()*
- *HAL_DCACHE_InvalidateByAddrCallback()*
- *HAL_DCACHE_CleanAndInvalidateByAddrCallback()*
- *HAL_DCACHE_InvalidateCompleteCallback()*
- *HAL_DCACHE_ErrorCallback()*
- *HAL_DCACHE_GetState()*
- *HAL_DCACHE_GetError()*

18.2.5 Peripheral State

This subsection permit to get in run-time the status of the peripheral and the data flow.
This section contains the following APIs:

- *HAL_DCACHE_GetState()*
- *HAL_DCACHE_GetError()*

18.2.6 Detailed description of functions

HAL_DCACHE_Init

Function name

HAL_StatusTypeDef HAL_DCACHE_Init (DCACHE_HandleTypeDef * hdcache)

Function description

Initializes the DCACHE according to the specified parameters in the DCACHE_InitTypeDef and initialize the associated handle.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHE.

Return values

- **HAL:** status

HAL_DCACHE_DeInit

Function name

HAL_StatusTypeDef HAL_DCACHE_DeInit (DCACHE_HandleTypeDef * hdcache)

Function description

DeInitialize the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

HAL_DCACHE_MspInit

Function name

void HAL_DCACHE_MspInit (DCACHE_HandleTypeDef * hdcache)

Function description

Initialize the DCACHE MSP.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_MspDeInit

Function name

void HAL_DCACHE_MspDeInit (DCACHE_HandleTypeDef * hdcache)

Function description

DeInitialize the DCACHE MSP.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_Enable

Function name

HAL_StatusTypeDef HAL_DCACHE_Enable (DCACHE_HandleTypeDef * hdcache)

Function description

Enable the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

HAL_DCACHE_Disable

Function name

HAL_StatusTypeDef HAL_DCACHE_Disable (DCACHE_HandleTypeDef * hdcache)

Function description

Disable the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

HAL_DCACHE_IsEnabled

Function name

uint32_t HAL_DCACHE_IsEnabled (const DCACHE_HandleTypeDef * hdcache)

Function description

Check whether the Data Cache is enabled or not.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Status:** (0: disabled, 1: enabled)

HAL_DCACHE_SetReadBurstType

Function name

HAL_StatusTypeDef HAL_DCACHE_SetReadBurstType (DCACHE_HandleTypeDef * hdcache, uint32_t ReadBurstType)

Function description

Set Read Burst Type.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **ReadBurstType:** Burst type to be applied for Data Cache DCACHE_READ_BURST_WRAP, DCACHE_READ_BURST_INC.

Return values

- **HAL:** status

HAL_DCACHE_Invalidate

Function name

HAL_StatusTypeDef HAL_DCACHE_Invalidate (DCACHE_HandleTypeDef * hdcache)

Function description

Invalidate the Data cache.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

Notes

- This function waits for end of full cache invalidation

HAL_DCACHE_InvalidateByAddr

Function name

HAL_StatusTypeDef HAL_DCACHE_InvalidateByAddr (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Invalidate the Data cache for a specific region.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Invalidated
- **dSize:** Size of the region to be Invalidated(in bytes)

Return values

- **HAL:** status

Notes

- This function waits for end of cache Invalidation

HAL_DCACHE_CleanByAddr

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanByAddr (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean the Data cache by Addr.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned
- **dSize:** Size of the region to be Cleaned (in bytes)

Return values

- **HAL:** status

Notes

- This function waits for end of cache Clean

HAL_DCACHE_CleanInvalidByAddr

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanInvalidByAddr (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean and Invalidate the Data cache by Addr.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned and Invalidated
- **dSize:** Size of the region to be Cleaned and Invalidated (in bytes)

Return values

- **HAL:** status

Notes

- This function waits for end of cache Clean and Invalidation

HAL_DCACHE_Invalidate_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_Invalidate_IT (DCACHE_HandleTypeDef * hdcache)

Function description

Invalidate the Data cache with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_InvalidateByAddr_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_InvalidateByAddr_IT (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Invalidate the Data cache by Addr with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Invalidated
- **dSize:** Size of the region to be Invalidated

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_CleanByAddr_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanByAddr_IT (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean the Data cache by Addr with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned
- **dSize:** Size of the region to be Cleaned

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_CleanInvalidByAddr_IT

Function name

HAL_StatusTypeDef HAL_DCACHE_CleanInvalidByAddr_IT (DCACHE_HandleTypeDef * hdcache, const uint32_t *const pAddr, uint32_t dSize)

Function description

Clean and Invalidate the Data cache by Addr with interrupt.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **pAddr:** Start address of the region to be Cleaned and Invalidated
- **dSize:** Size of the region to be Cleaned and Invalidated

Return values

- **HAL:** status

Notes

- This function launches maintenance operation and returns immediately. User application shall resort to interrupt generation to check the end of operation.

HAL_DCACHE_IRQHandler

Function name

void HAL_DCACHE_IRQHandler (DCACHE_HandleTypeDef * hdcache)

Function description

Handle the Data Cache interrupt request.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

Notes

- This API should be called under the DCACHEx_IRQHandler().

HAL_DCACHE_ErrorCallback

Function name

void HAL_DCACHE_ErrorCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Error callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_CleanByAddrCallback

Function name

void HAL_DCACHE_CleanByAddrCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache clean command by address callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_InvalidateByAddrCallback

Function name

void HAL_DCACHE_InvalidateByAddrCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache Invalidate command by address callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_InvalidateCompleteCallback

Function name

void HAL_DCACHE_InvalidateCompleteCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache full invalidation complete callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_CleanAndInvalidateByAddrCallback

Function name

void HAL_DCACHE_CleanAndInvalidateByAddrCallback (DCACHE_HandleTypeDef * hdcache)

Function description

Cache clean and Invalidate command by address callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **None:**

HAL_DCACHE_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCACHE_RegisterCallback (DCACHE_HandleTypeDef * hdcache, HAL_DCACHE_CallbackIDTypeDef CallbackID, pDCACHE_CallbackTypeDef pCallback)

Function description

Register a User DCACHE Callback To be used instead of the weak predefined callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCACHE_CLEAN_BY_ADDRESS_CB_ID Clean By Addr callback ID
  - HAL_DCACHE_INVALIDATE_BY_ADDRESS_CB_ID Invalidate By Addr callback ID
  - HAL_DCACHE_CLEAN_AND_INVALIDATE_BY_ADDRESS_CB_ID Clean and Invalidate By Addr callback ID
  - HAL_DCACHE_INVALIDATE_COMPLETE_CB_ID Invalidate Complete ID
  - HAL_DCACHE_ERROR_CB_ID Error callback ID
  - HAL_DCACHE_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCACHE_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_DCACHE_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCACHE_UnRegisterCallback (DCACHE_HandleTypeDef * hdcache, HAL_DCACHE_CallbackIDTypeDef CallbackID)

Function description

Unregister an DCACHE Callback DCACHE callback is redirected to the weak predefined callback.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_DCACHE_CLEAN_BY_ADDRESS_CB_ID Clean By Addr callback ID
  – HAL_DCACHE_INVALIDATE_BY_ADDRESS_CB_ID Invalidate By Addr callback ID
  – HAL_DCACHE_CLEAN_AND_INVALIDATE_BY_ADDRESS_CB_ID Clean and Invalidate By Addr callback ID
  – HAL_DCACHE_INVALIDATE_COMPLETE_CB_ID Invalidate Complete callback ID
  – HAL_DCACHE_ERROR_CB_ID Error callback ID
  – HAL_DCACHE_MSPINIT_CB_ID MspInit callback ID
  – HAL_DCACHE_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_DCACHE_Monitor_GetReadHitValue

Function name

uint32_t HAL_DCACHE_Monitor_GetReadHitValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Read Hit monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Read:** Hit monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_GetReadMissValue

Function name

uint32_t HAL_DCACHE_Monitor_GetReadMissValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Read Miss monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Read:** Miss monitoring value

Notes

- Upon reaching the 16-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_GetWriteHitValue

Function name

uint32_t HAL_DCACHE_Monitor_GetWriteHitValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Write Hit monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Write:** Hit monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_GetWriteMissValue

Function name

uint32_t HAL_DCACHE_Monitor_GetWriteMissValue (const DCACHE_HandleTypeDef * hdcache)

Function description

Get the Data Cache performance Write Miss monitoring value.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **Write:** Miss monitoring value

Notes

- Upon reaching the 16-bit maximum value, monitor does not wrap.

HAL_DCACHE_Monitor_Reset

Function name

HAL_StatusTypeDef HAL_DCACHE_Monitor_Reset (DCACHE_HandleTypeDef * hdcache, uint32_t MonitorType)

Function description

Reset the Data Cache performance monitoring values.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **MonitorType:** Monitoring type This parameter can be a combination of the following values:
  – DCACHE_MONITOR_READ_HIT
  – DCACHE_MONITOR_READ_MISS
  – DCACHE_MONITOR_WRITE_HIT
  – DCACHE_MONITOR_WRITE_MISS
  – DCACHE_MONITOR_ALL

Return values

- **HAL:** status

HAL_DCACHE_Monitor_Start

Function name

HAL_StatusTypeDef HAL_DCACHE_Monitor_Start (DCACHE_HandleTypeDef * hdcache, uint32_t MonitorType)

Function description

Start the Data Cache performance monitoring.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **MonitorType:** Monitoring type This parameter can be a combination of the following values:
  – DCACHE_MONITOR_READ_HIT
  – DCACHE_MONITOR_READ_MISS
  – DCACHE_MONITOR_WRITE_HIT
  – DCACHE_MONITOR_WRITE_MISS
  – DCACHE_MONITOR_ALL

Return values

- **HAL:** status

HAL_DCACHE_Monitor_Stop

Function name

HAL_StatusTypeDef HAL_DCACHE_Monitor_Stop (DCACHE_HandleTypeDef * hdcache, uint32_t MonitorType)

Function description

Stop the Data Cache performance monitoring.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.
- **MonitorType:** Monitoring type This parameter can be a combination of the following values:
  – DCACHE_MONITOR_READ_HIT
  – DCACHE_MONITOR_READ_MISS
  – DCACHE_MONITOR_WRITE_HIT
  – DCACHE_MONITOR_WRITE_MISS
  – DCACHE_MONITOR_ALL

Return values

- **HAL:** status

Notes

- Stopping the monitoring does not reset the values.

HAL_DCACHE_GetState

Function name

HAL_DCACHE_StateTypeDef HAL_DCACHE_GetState (const DCACHE_HandleTypeDef * hdcache)

Function description

Return the DCACHE handle state.

Parameters

- **hdcache:** Pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHEx peripheral.

Return values

- **HAL:** state

HAL_DCACHE_GetError

Function name

uint32_t HAL_DCACHE_GetError (const DCACHE_HandleTypeDef * hdcache)

Function description

Return the DCACHE error code.

Parameters

- **hdcache:** pointer to a DCACHE_HandleTypeDef structure that contains the configuration information for the specified DCACHE.

Return values

- **DCACHE:** Error Code

18.3 DCACHE Firmware driver defines

The following section lists the various define and macros of the module.

18.3.1 DCACHE

DCACHE

Command Operation

LL_DCACHE_COMMAND_NO_OPERATION

LL_DCACHE_COMMAND_CLEAN_BY_ADDR

LL_DCACHE_COMMAND_INVALIDATE_BY_ADDR

LL_DCACHE_COMMAND_CLEAN_INVALIDATE_BY_ADDR

DCACHE Error Code

HAL_DCACHE_ERROR_NONE

No error

HAL_DCACHE_ERROR_TIMEOUT

Timeout error

HAL_DCACHE_ERROR_INVALID_CALLBACK

Invalid callback error

HAL_DCACHE_ERROR_EVICTION_CLEAN

Eviction or clean operation write-back error

HAL_DCACHE_ERROR_INVALID_OPERATION

Invalid operation

DCACHE Exported Macros

__HAL_DCACHE_ENABLE_IT

Description:

- Enable DCACHE interrupts.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __INTERRUPT__: specifies the DCACHE interrupt sources to be enabled. This parameter can be any combination of the following values:
  – DCACHE_IT_BUSYEND Busy end interrupt
  – DCACHE_IT_ERROR Cache error interrupt
  – DCACHE_IT_CMDEND Cache Command end interrupt

Return value:

- None

__HAL_DCACHE_DISABLE_IT

Description:

- Disable DCACHE interrupts.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __INTERRUPT__: specifies the DCACHE interrupt sources to be disabled. This parameter can be any combination of the following values:
  – DCACHE_IT_BUSYEND Busy end interrupt
  – DCACHE_IT_ERROR Cache error interrupt
  – DCACHE_IT_CMDEND Cache Command end interrupt

Return value:

- None

__HAL_DCACHE_GET_IT_SOURCE

Description:

- Check whether the specified DCACHE interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __INTERRUPT__: specifies the DCACHE interrupt source to check. This parameter can be any combination of the following values:
  – DCACHE_IT_BUSYEND Busy end interrupt
  – DCACHE_IT_ERROR Cache error interrupt
  – DCACHE_IT_CMDEND Cache Command end interrupt

Return value:

- The: state of __INTERRUPT__ (SET or RESET).

__HAL_DCACHE_GET_FLAG

Description:

- Check whether the selected DCACHE flag is set or not.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - DCACHE_FLAG_BUSY Busy flag
  - DCACHE_FLAG_BUSYEND Busy end flag
  - DCACHE_FLAG_ERROR Cache error flag
  - DCACHE_FLAG_BUSYCMD Cache Busy command flag
  - DCACHE_FLAG_CMDEND Cache command end flag

Return value:

- The: state of __FLAG__ (0 or 1).

__HAL_DCACHE_CLEAR_FLAG

Description:

- Clear the selected DCACHE flags.

Parameters:

- __HANDLE__: specifies the DCACHE handle.
- __FLAG__: specifies the DCACHE flags to clear. This parameter can be any combination of the following values:
  - DCACHE_FLAG_BUSYEND Busy end flag
  - DCACHE_FLAG_ERROR Cache error flag
  - DCACHE_FLAG_CMDEND Cache command end flag

Flags

DCACHE_FLAG_BUSY

Busy flag

DCACHE_FLAG_BUSYEND

Busy end flag

DCACHE_FLAG_ERROR

Cache error flag

DCACHE_FLAG_BUSYCMD

Busy command flag

DCACHE_FLAG_CMDEND

Command end flag

Interrupts

DCACHE_IT_BUSYEND

Busy end interrupt

DCACHE_IT_ERROR

Cache error interrupt

DCACHE_IT_CMDEND

Command end interrupt

Monitor type

DCACHE_MONITOR_READ_HIT

Read Hit monitoring

DCACHE_MONITOR_READ_MISS

Read Miss monitoring

DCACHE_MONITOR_WRITE_HIT

Write Hit monitoring

DCACHE_MONITOR_WRITE_MISS

Write Miss monitoring

DCACHE_MONITOR_ALL

LL_DCACHE_MONITOR_READ_HIT

Read Hit monitoring

LL_DCACHE_MONITOR_READ_MISS

Read Miss monitoring

LL_DCACHE_MONITOR_WRITE_HIT

Write Hit monitoring

LL_DCACHE_MONITOR_WRITE_MISS

Write Miss monitoring

LL_DCACHE_MONITOR_ALL

Remapped Output burst type

DCACHE_READ_BURST_WRAP

WRAP

DCACHE_READ_BURST_INCR

INCR

LL_DCACHE_READ_BURST_WRAP

WRAP

LL_DCACHE_READ_BURST_INCR

INCR

19 HAL DCMI Generic Driver

19.1 DCMI Firmware driver registers structures

19.1.1 DCMI_CodesInitTypeDef

DCMI_CodesInitTypeDef is defined in the stm32u5xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartCode***
- ***uint8_t LineStartCode***
- ***uint8_t LineEndCode***
- ***uint8_t FrameEndCode***

Field Documentation

- ***uint8_t DCMI_CodesInitTypeDef::FrameStartCode***
  Specifies the code of the frame start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineStartCode***
  Specifies the code of the line start delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::LineEndCode***
  Specifies the code of the line end delimiter.
- ***uint8_t DCMI_CodesInitTypeDef::FrameEndCode***
  Specifies the code of the frame end delimiter.

19.1.2 DCMI_SyncUnmaskTypeDef

DCMI_SyncUnmaskTypeDef is defined in the stm32u5xx_hal_dcmi.h

Data Fields

- ***uint8_t FrameStartUnmask***
- ***uint8_t LineStartUnmask***
- ***uint8_t LineEndUnmask***
- ***uint8_t FrameEndUnmask***

Field Documentation

- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameStartUnmask***
  Specifies the frame start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineStartUnmask***
  Specifies the line start delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::LineEndUnmask***
  Specifies the line end delimiter unmask.
- ***uint8_t DCMI_SyncUnmaskTypeDef::FrameEndUnmask***
  Specifies the frame end delimiter unmask.

19.1.3 DCMI_InitTypeDef

DCMI_InitTypeDef is defined in the stm32u5xx_hal_dcmi.h

Data Fields

- ***uint32_t SynchroMode***
- ***uint32_t PCKPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t CaptureRate***
- ***uint32_t ExtendedDataMode***
- ***DCMI_CodesInitTypeDef SyncroCode***
- ***uint32_t JPEGMode***
- ***uint32_t ByteSelectMode***

- *uint32_t ByteSelectStart*
- *uint32_t LineSelectMode*
- *uint32_t LineSelectStart*

Field Documentation

- ***uint32_t DCMI_InitTypeDef::SynchroMode***
  Specifies the Synchronization Mode: Hardware or Embedded. This parameter can be a value of *DCMI_Synchronization_Mode*
- ***uint32_t DCMI_InitTypeDef::PCKPolarity***
  Specifies the Pixel clock polarity: Falling or Rising. This parameter can be a value of *DCMI_PIXCK_Polarity*
- ***uint32_t DCMI_InitTypeDef::VSPolarity***
  Specifies the Vertical synchronization polarity: High or Low. This parameter can be a value of *DCMI_VSYNC_Polarity*
- ***uint32_t DCMI_InitTypeDef::HSPolarity***
  Specifies the Horizontal synchronization polarity: High or Low. This parameter can be a value of *DCMI_HSYNC_Polarity*
- ***uint32_t DCMI_InitTypeDef::CaptureRate***
  Specifies the frequency of frame capture: All, 1/2 or 1/4. This parameter can be a value of *DCMI_Capture_Rate*
- ***uint32_t DCMI_InitTypeDef::ExtendedDataMode***
  Specifies the data width: 8-bit, 10-bit, 12-bit or 14-bit. This parameter can be a value of *DCMI_Extended_Data_Mode*
- ***DCMI_CodesInitTypeDef DCMI_InitTypeDef::SyncroCode***
  Specifies the code of the line/frame start delimiter and the line/frame end delimiter
- ***uint32_t DCMI_InitTypeDef::JPEGMode***
  Enable or Disable the JPEG mode. This parameter can be a value of *DCMI_MODE_JPEG*
- ***uint32_t DCMI_InitTypeDef::ByteSelectMode***
  Specifies the data to be captured by the interface This parameter can be a value of *DCMI_Byte_Select_Mode*
- ***uint32_t DCMI_InitTypeDef::ByteSelectStart***
  Specifies if the data to be captured by the interface is even or odd This parameter can be a value of *DCMI_Byte_Select_Start*
- ***uint32_t DCMI_InitTypeDef::LineSelectMode***
  Specifies the line of data to be captured by the interface This parameter can be a value of *DCMI_Line_Select_Mode*
- ***uint32_t DCMI_InitTypeDef::LineSelectStart***
  Specifies if the line of data to be captured by the interface is even or odd This parameter can be a value of *DCMI_Line_Select_Start*

19.1.4 __DCMI_HandleTypeDef

__DCMI_HandleTypeDef is defined in the stm32u5xx_hal_dcmi.h

Data Fields

- *DCMI_TypeDef * Instance*
- *DCMI_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_DCMI_StateTypeDef State*
- *__IO uint32_t XferCount*
- *__IO uint32_t XferSize*
- *uint32_t XferTransferNumber*
- *uint32_t pBuffPtr*
- *DMA_HandleTypeDef * DMA_Handle*
- *__IO uint32_t ErrorCode*
- *void(* FrameEventCallback*
- *void(* VsyncEventCallback*

- ***void(* LineEventCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DCMI_TypeDef* __DCMI_HandleTypeDef::Instance***
  DCMI Register base address
- ***DCMI_InitTypeDef __DCMI_HandleTypeDef::Init***
  DCMI parameters
- ***HAL_LockTypeDef __DCMI_HandleTypeDef::Lock***
  DCMI locking object
- ***__IO HAL_DCMI_StateTypeDef __DCMI_HandleTypeDef::State***
  DCMI state
- ***__IO uint32_t __DCMI_HandleTypeDef::XferCount***
  DMA transfer counter
- ***__IO uint32_t __DCMI_HandleTypeDef::XferSize***
  DMA transfer size
- ***uint32_t __DCMI_HandleTypeDef::XferTransferNumber***
  DMA transfer number
- ***uint32_t __DCMI_HandleTypeDef::pBuffPtr***
  Pointer to DMA output buffer
- ***DMA_HandleTypeDef* __DCMI_HandleTypeDef::DMA_Handle***
  Pointer to the DMA handler
- ***__IO uint32_t __DCMI_HandleTypeDef::ErrorCode***
  DCMI Error code
- ***void(* __DCMI_HandleTypeDef::FrameEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Frame Event Callback
- ***void(* __DCMI_HandleTypeDef::VsyncEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Vsync Event Callback
- ***void(* __DCMI_HandleTypeDef::LineEventCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Line Event Callback
- ***void(* __DCMI_HandleTypeDef::ErrorCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Error Callback
- ***void(* __DCMI_HandleTypeDef::MspInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp Init callback
- ***void(* __DCMI_HandleTypeDef::MspDeInitCallback)(struct __DCMI_HandleTypeDef *hdcmi)***
  DCMI Msp DeInit callback

19.2 DCMI Firmware driver API description

The following section lists the various functions of the DCMI library.

19.2.1 How to use this driver

The sequence below describes how to use this driver to capture image from a camera module connected to the DCMI Interface. This sequence does not take into account the configuration of the camera module, which should be made before to configure and enable the DCMI to capture images.

1. Program the required configuration through following parameters: horizontal and vertical polarity, pixel clock polarity, Capture Rate, Synchronization Mode, code of the frame delimiter and data width using HAL_DCMI_Init() function.
2. Configure the selected DMA stream to transfer Data from DCMI DR register to the destination memory buffer.
3. Program the required configuration through following parameters: DCMI mode, destination memory Buffer address and the data length and enable capture using HAL_DCMI_Start_DMA() function.
4. Optionally, configure and Enable the CROP feature to select a rectangular window from the received image using HAL_DCMI_ConfigCrop() and HAL_DCMI_EnableCrop() functions
5. The capture can be stopped using HAL_DCMI_Stop() function.

6. To control DCMI state you can use the function HAL_DCMI_GetState().

DCMI HAL driver macros list

Below the list of most used macros in DCMI HAL driver.

- __HAL_DCMI_ENABLE: Enable the DCMI peripheral.
- __HAL_DCMI_DISABLE: Disable the DCMI peripheral.
- __HAL_DCMI_GET_FLAG: Get the DCMI pending flags.
- __HAL_DCMI_CLEAR_FLAG: Clear the DCMI pending flags.
- __HAL_DCMI_ENABLE_IT: Enable the specified DCMI interrupts.
- __HAL_DCMI_DISABLE_IT: Disable the specified DCMI interrupts.
- __HAL_DCMI_GET_IT_SOURCE: Check whether the specified DCMI interrupt has occurred or not.

Note: *You can refer to the DCMI HAL driver header file for more useful macros*

Callback registration

19.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DCMI
- De-initialize the DCMI

This section contains the following APIs:

- *HAL_DCMI_Init()*
- *HAL_DCMI_DeInit()*
- *HAL_DCMI_MspInit()*
- *HAL_DCMI_MspDeInit()*
- *HAL_DCMI_RegisterCallback()*
- *HAL_DCMI_UnRegisterCallback()*

19.2.3 IO operation functions

This section provides functions allowing to:

- Configure destination address and data length and Enables DCMI DMA request and enables DCMI capture
- Stop the DCMI capture.
- Handles DCMI interrupt request.

This section contains the following APIs:

- *HAL_DCMI_Start_DMA()*
- *HAL_DCMI_Stop()*
- *HAL_DCMI_Suspend()*
- *HAL_DCMI_Resume()*
- *HAL_DCMI_IRQHandler()*
- *HAL_DCMI_ErrorCallback()*
- *HAL_DCMI_LineEventCallback()*
- *HAL_DCMI_VsyncEventCallback()*
- *HAL_DCMI_FrameEventCallback()*

19.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the CROP feature.
- Enable/Disable the CROP feature.
- Set embedded synchronization delimiters unmasks.

This section contains the following APIs:

- *HAL_DCMI_ConfigCrop()*
- *HAL_DCMI_DisableCrop()*
- *HAL_DCMI_EnableCrop()*
- *HAL_DCMI_ConfigSyncUnmask()*

19.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to
- Check the DCMI state.
- Get the specific DCMI error flag.

This section contains the following APIs:
- *HAL_DCMI_GetState()*
- *HAL_DCMI_GetError()*
- *HAL_DCMI_RegisterCallback()*
- *HAL_DCMI_UnRegisterCallback()*

19.2.6 Detailed description of functions

HAL_DCMI_Init

Function name

HAL_StatusTypeDef HAL_DCMI_Init (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI according to the specified parameters in the DCMI_InitTypeDef and create the associated handle.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DeInit

Function name

HAL_StatusTypeDef HAL_DCMI_DeInit (DCMI_HandleTypeDef * hdcmi)

Function description

Deinitializes the DCMI peripheral registers to their default reset values.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_MspInit

Function name

void HAL_DCMI_MspInit (DCMI_HandleTypeDef * hdcmi)

Function description

Initializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_MspDeInit

Function name

void HAL_DCMI_MspDeInit (DCMI_HandleTypeDef * hdcmi)

Function description

DeInitializes the DCMI MSP.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_RegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID, pDCMI_CallbackTypeDef pCallback)

Function description

Register a User DCMI Callback To be used instead of the weak predefined callback.

Parameters

- **hdcmi:** DCMI handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCMI_LINE_EVENT_CB_ID Line Event callback ID
  - HAL_DCMI_FRAME_EVENT_CB_ID Frame Event callback ID
  - HAL_DCMI_VSYNC_EVENT_CB_ID Vsync Event callback ID
  - HAL_DCMI_ERROR_CB_ID Error callback ID
  - HAL_DCMI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCMI_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_DCMI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DCMI_UnRegisterCallback (DCMI_HandleTypeDef * hdcmi, HAL_DCMI_CallbackIDTypeDef CallbackID)

Function description

Unregister a DCMI Callback DCMI callback is redirected to the weak predefined callback.

Parameters

- **hdcmi:** DCMI handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_DCMI_LINE_EVENT_CB_ID Line Event callback ID
  - HAL_DCMI_FRAME_EVENT_CB_ID Frame Event callback ID
  - HAL_DCMI_VSYNC_EVENT_CB_ID Vsync Event callback ID
  - HAL_DCMI_ERROR_CB_ID Error callback ID
  - HAL_DCMI_MSPINIT_CB_ID MspInit callback ID
  - HAL_DCMI_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_DCMI_Start_DMA

Function name

HAL_StatusTypeDef HAL_DCMI_Start_DMA (DCMI_HandleTypeDef * hdcmi, uint32_t DCMI_Mode, uint32_t pData, uint32_t Length)

Function description

Enables DCMI DMA request and enables DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **DCMI_Mode:** DCMI capture mode snapshot or continuous grab.
- **pData:** The destination memory Buffer address (LCD Frame buffer).
- **Length:** The length of capture to be transferred.

Return values

- **HAL:** status

HAL_DCMI_Stop

Function name

HAL_StatusTypeDef HAL_DCMI_Stop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable DCMI DMA request and Disable DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Suspend

Function name

HAL_StatusTypeDef HAL_DCMI_Suspend (DCMI_HandleTypeDef * hdcmi)

Function description

Suspend DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_Resume

Function name

HAL_StatusTypeDef HAL_DCMI_Resume (DCMI_HandleTypeDef * hdcmi)

Function description

Resume DCMI capture.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ErrorCallback

Function name

void HAL_DCMI_ErrorCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Error DCMI callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_LineEventCallback

Function name

void HAL_DCMI_LineEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Line Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_FrameEventCallback

Function name

void HAL_DCMI_FrameEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

Frame Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_VsyncEventCallback

Function name

void HAL_DCMI_VsyncEventCallback (DCMI_HandleTypeDef * hdcmi)

Function description

VSYNC Event callback.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **None:**

HAL_DCMI_IRQHandler

Function name

void HAL_DCMI_IRQHandler (DCMI_HandleTypeDef * hdcmi)

Function description

Handles DCMI interrupt request.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for the DCMI.

Return values

- **None:**

HAL_DCMI_ConfigCrop

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigCrop (DCMI_HandleTypeDef * hdcmi, uint32_t X0, uint32_t Y0, uint32_t XSize, uint32_t YSize)

Function description

Configure the DCMI CROP coordinate.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **YSize:** DCMI Line number
- **XSize:** DCMI Pixel per line
- **X0:** DCMI window X offset
- **Y0:** DCMI window Y offset

Return values

- **HAL:** status

HAL_DCMI_EnableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_EnableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Enable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_DisableCrop

Function name

HAL_StatusTypeDef HAL_DCMI_DisableCrop (DCMI_HandleTypeDef * hdcmi)

Function description

Disable the Crop feature.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** status

HAL_DCMI_ConfigSyncUnmask

Function name

HAL_StatusTypeDef HAL_DCMI_ConfigSyncUnmask (DCMI_HandleTypeDef * hdcmi, DCMI_SyncUnmaskTypeDef * SyncUnmask)

Function description

Set embedded synchronization delimiters unmasks.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.
- **SyncUnmask:** pointer to a DCMI_SyncUnmaskTypeDef structure that contains the embedded synchronization delimiters unmasks.

Return values

- **HAL:** status

HAL_DCMI_GetState

Function name

HAL_DCMI_StateTypeDef HAL_DCMI_GetState (const DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI state.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **HAL:** state

HAL_DCMI_GetError

Function name

uint32_t HAL_DCMI_GetError (const DCMI_HandleTypeDef * hdcmi)

Function description

Return the DCMI error code.

Parameters

- **hdcmi:** pointer to a DCMI_HandleTypeDef structure that contains the configuration information for DCMI.

Return values

- **DCMI:** Error Code

19.3 DCMI Firmware driver defines

The following section lists the various define and macros of the module.

19.3.1 DCMI

DCMI

DCMI Byte Select Mode

DCMI_BSM_ALL

Interface captures all received data

DCMI_BSM_OTHER

Interface captures every other byte from the received data

DCMI_BSM_ALTERNATE_4

Interface captures one byte out of four

DCMI_BSM_ALTERNATE_2

Interface captures two bytes out of four

DCMI Byte Select Start

DCMI_OEBS_ODD

Interface captures first data from the frame/line start, second one being dropped

DCMI_OEBS_EVEN

Interface captures second data from the frame/line start, first one being dropped

DCMI Capture Mode

DCMI_MODE_CONTINUOUS

The received data are transferred continuously into the destination memory through the DMA

DCMI_MODE_SNAPSHOT

Once activated, the interface waits for the start of frame and then transfers a single frame through the DMA

DCMI Capture Rate

DCMI_CR_ALL_FRAME

All frames are captured

DCMI_CR_ALTERNATE_2_FRAME

Every alternate frame captured

DCMI_CR_ALTERNATE_4_FRAME

One frame in 4 frames captured

DCMI Error Code

HAL_DCMI_ERROR_NONE

No error

HAL_DCMI_ERROR_OVR

Overrun error

HAL_DCMI_ERROR_SYNC

Synchronization error

HAL_DCMI_ERROR_TIMEOUT

Timeout error

HAL_DCMI_ERROR_DMA

DMA error

HAL_DCMI_ERROR_INVALID_CALLBACK

Invalid callback error

DCMI Exported Macros

__HAL_DCMI_RESET_HANDLE_STATE

Description:
- Reset DCMI handle state.

Parameters:
- __HANDLE__: specifies the DCMI handle.

Return value:
- None

__HAL_DCMI_ENABLE

Description:
- Enable the DCMI.

Parameters:
- __HANDLE__: DCMI handle

Return value:
- None

__HAL_DCMI_DISABLE

Description:
- Disable the DCMI.

Parameters:
- __HANDLE__: DCMI handle

Return value:
- None

__HAL_DCMI_GET_FLAG

Description:

- Get the DCMI pending flag.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: Get the specified flag. This parameter can be one of the following values (no combination allowed)
  - DCMI_FLAG_HSYNC: HSYNC pin state (active line / synchronization between lines)
  - DCMI_FLAG_VSYNC: VSYNC pin state (active frame / synchronization between frames)
  - DCMI_FLAG_FNE: FIFO empty flag
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVRRI: Overrun flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask
  - DCMI_FLAG_FRAMEMI: DCMI Capture complete masked interrupt status
  - DCMI_FLAG_OVRMI: DCMI Overrun masked interrupt status
  - DCMI_FLAG_ERRMI: DCMI Synchronization error masked interrupt status
  - DCMI_FLAG_VSYNCMI: DCMI VSYNC masked interrupt status
  - DCMI_FLAG_LINEMI: DCMI Line masked interrupt status

Return value:

- The: state of FLAG.

__HAL_DCMI_CLEAR_FLAG

Description:

- Clear the DCMI pending flags.

Parameters:

- __HANDLE__: DCMI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DCMI_FLAG_FRAMERI: Frame capture complete flag mask
  - DCMI_FLAG_OVFRI: Overflow flag mask
  - DCMI_FLAG_ERRRI: Synchronization error flag mask
  - DCMI_FLAG_VSYNCRI: VSYNC flag mask
  - DCMI_FLAG_LINERI: Line flag mask

Return value:

- None

__HAL_DCMI_ENABLE_IT

Description:

- Enable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_DISABLE_IT

Description:

- Disable the specified DCMI interrupts.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- None

__HAL_DCMI_GET_IT_SOURCE

Description:

- Check whether the specified DCMI interrupt has occurred or not.

Parameters:

- __HANDLE__: DCMI handle
- __INTERRUPT__: specifies the DCMI interrupt source to check. This parameter can be one of the following values:
  - DCMI_IT_FRAME: Frame capture complete interrupt mask
  - DCMI_IT_OVF: Overflow interrupt mask
  - DCMI_IT_ERR: Synchronization error interrupt mask
  - DCMI_IT_VSYNC: VSYNC interrupt mask
  - DCMI_IT_LINE: Line interrupt mask

Return value:

- The: state of INTERRUPT.

DCMI Extended Data Mode

DCMI_EXTEND_DATA_8B

Interface captures 8-bit data on every pixel clock

DCMI_EXTEND_DATA_10B

Interface captures 10-bit data on every pixel clock

DCMI_EXTEND_DATA_12B

Interface captures 12-bit data on every pixel clock

DCMI_EXTEND_DATA_14B

Interface captures 14-bit data on every pixel clock

DCMI Flags

DCMI_FLAG_HSYNC

HSYNC pin state (active line / synchronization between lines)

DCMI_FLAG_VSYNC

VSYNC pin state (active frame / synchronization between frames)

DCMI_FLAG_FNE

FIFO not empty flag

DCMI_FLAG_FRAMERI

Frame capture complete interrupt flag

DCMI_FLAG_OVRRI

Overrun interrupt flag

DCMI_FLAG_ERRRI

Synchronization error interrupt flag

DCMI_FLAG_VSYNCRI

VSYNC interrupt flag

DCMI_FLAG_LINERI

Line interrupt flag

DCMI_FLAG_FRAMEMI

DCMI Frame capture complete masked interrupt status

DCMI_FLAG_OVRMI

DCMI Overrun masked interrupt status

DCMI_FLAG_ERRMI

DCMI Synchronization error masked interrupt status

DCMI_FLAG_VSYNCMI

DCMI VSYNC masked interrupt status

DCMI_FLAG_LINEMI

DCMI Line masked interrupt status

DCMI HSYNC Polarity

DCMI_HSPOLARITY_LOW

Horizontal synchronization active Low

DCMI_HSPOLARITY_HIGH

Horizontal synchronization active High

DCMI interrupt sources

DCMI_IT_FRAME

Capture complete interrupt

DCMI_IT_OVR

Overrun interrupt

DCMI_IT_ERR

Synchronization error interrupt

DCMI_IT_VSYNC

VSYNC interrupt

DCMI_IT_LINE

Line interrupt

DCMI Line Select Mode

DCMI_LSM_ALL

Interface captures all received lines

DCMI_LSM_ALTERNATE_2

Interface captures one line out of two

DCMI Line Select Start

DCMI_OELS_ODD

Interface captures first line from the frame start, second one being dropped

DCMI_OELS_EVEN

Interface captures second line from the frame start, first one being dropped

DCMI Mis Index

DCMI_MIS_INDEX

DCMI MIS register index

DCMI MODE JPEG

DCMI_JPEG_DISABLE

Mode JPEG Disabled

DCMI_JPEG_ENABLE

Mode JPEG Enabled

DCMI PIXCK Polarity

DCMI_PCKPOLARITY_FALLING

Pixel clock active on Falling edge

DCMI_PCKPOLARITY_RISING

Pixel clock active on Rising edge

DCMI SR Index

DCMI_SR_INDEX

DCMI SR register index

DCMI Stop Time Out

HAL_TIMEOUT_DCMI_STOP

DCMI Synchronization Mode

DCMI_SYNCHRO_HARDWARE

Hardware synchronization data capture (frame/line start/stop) is synchronized with the HSYNC/VSYNC signals

DCMI_SYNCHRO_EMBEDDED

Embedded synchronization data capture is synchronized with synchronization codes embedded in the data flow

DCMI VSYNC Polarity

DCMI_VSPOLARITY_LOW

Vertical synchronization active Low

DCMI_VSPOLARITY_HIGH

Vertical synchronization active High

DCMI Window Coordinate

DCMI_WINDOW_COORDINATE

Window coordinate

DCMI Window Height

DCMI_WINDOW_HEIGHT

Window Height

20 HAL DMA2D Generic Driver

20.1 DMA2D Firmware driver registers structures

20.1.1 DMA2D_CLUTCfgTypeDef

DMA2D_CLUTCfgTypeDef is defined in the stm32u5xx_hal_dma2d.h

Data Fields

- ***uint32_t * pCLUT***
- ***uint32_t CLUTColorMode***
- ***uint32_t Size***

Field Documentation

- ***uint32_t* DMA2D_CLUTCfgTypeDef::pCLUT***
  Configures the DMA2D CLUT memory address.
- ***uint32_t DMA2D_CLUTCfgTypeDef::CLUTColorMode***
  Configures the DMA2D CLUT color mode. This parameter can be one value of ***DMA2D_CLUT_CM***.
- ***uint32_t DMA2D_CLUTCfgTypeDef::Size***
  Configures the DMA2D CLUT size. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

20.1.2 DMA2D_InitTypeDef

DMA2D_InitTypeDef is defined in the stm32u5xx_hal_dma2d.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputOffset***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t BytesSwap***
- ***uint32_t LineOffsetMode***

Field Documentation

- ***uint32_t DMA2D_InitTypeDef::Mode***
  Configures the DMA2D transfer mode. This parameter can be one value of ***DMA2D_Mode***.
- ***uint32_t DMA2D_InitTypeDef::ColorMode***
  Configures the color format of the output image. This parameter can be one value of ***DMA2D_Output_Color_Mode***.
- ***uint32_t DMA2D_InitTypeDef::OutputOffset***
  Specifies the Offset value. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_InitTypeDef::AlphaInverted***
  Select regular or inverted alpha value for the output pixel format converter. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_InitTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR) for the output pixel format converter. This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_InitTypeDef::BytesSwap***
  Select byte regular mode or bytes swap mode (two by two). This parameter can be one value of ***DMA2D_Bytes_Swap***.
- ***uint32_t DMA2D_InitTypeDef::LineOffsetMode***
  Configures how is expressed the line offset for the foreground, background and output. This parameter can be one value of ***DMA2D_Line_Offset_Mode***.

20.1.3 DMA2D_LayerCfgTypeDef

DMA2D_LayerCfgTypeDef is defined in the stm32u5xx_hal_dma2d.h

Data Fields

- ***uint32_t InputOffset***
- ***uint32_t InputColorMode***
- ***uint32_t AlphaMode***
- ***uint32_t InputAlpha***
- ***uint32_t AlphaInverted***
- ***uint32_t RedBlueSwap***
- ***uint32_t ChromaSubSampling***

Field Documentation

- ***uint32_t DMA2D_LayerCfgTypeDef::InputOffset***
  Configures the DMA2D foreground or background offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputColorMode***
  Configures the DMA2D foreground or background color mode. This parameter can be one value of ***DMA2D_Input_Color_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaMode***
  Configures the DMA2D foreground or background alpha mode. This parameter can be one value of ***DMA2D_Alpha_Mode***.
- ***uint32_t DMA2D_LayerCfgTypeDef::InputAlpha***
  Specifies the DMA2D foreground or background alpha value and color value in case of A8 or A4 color mode. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF except for the color modes detailed below.
  Note:
  – In case of A8 or A4 color mode (ARGB), this parameter must be a number between Min_Data = 0x00000000 and Max_Data = 0xFFFFFFFF where
    - InputAlpha[24:31] is the alpha value ALPHA[0:7]
    - InputAlpha[16:23] is the red value RED[0:7]
    - InputAlpha[8:15] is the green value GREEN[0:7]
    - InputAlpha[0:7] is the blue value BLUE[0:7].
- ***uint32_t DMA2D_LayerCfgTypeDef::AlphaInverted***
  Select regular or inverted alpha value. This parameter can be one value of ***DMA2D_Alpha_Inverted***.
- ***uint32_t DMA2D_LayerCfgTypeDef::RedBlueSwap***
  Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR). This parameter can be one value of ***DMA2D_RB_Swap***.
- ***uint32_t DMA2D_LayerCfgTypeDef::ChromaSubSampling***
  Configure the chroma sub-sampling mode for the YCbCr color mode This parameter can be one value of ***DMA2D_Chroma_Sub_Sampling***

20.1.4 __DMA2D_HandleTypeDef

__DMA2D_HandleTypeDef is defined in the stm32u5xx_hal_dma2d.h

Data Fields

- ***DMA2D_TypeDef * Instance***
- ***DMA2D_InitTypeDef Init***
- ***void(* XferCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* LineEventCallback***
- ***void(* CLUTLoadingCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***DMA2D_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***

- ***__IO HAL_DMA2D_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DMA2D_TypeDef* __DMA2D_HandleTypeDef::Instance***
  DMA2D register base address.
- ***DMA2D_InitTypeDef __DMA2D_HandleTypeDef::Init***
  DMA2D communication parameters.
- ***void(* __DMA2D_HandleTypeDef::XferCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer complete callback.
- ***void(* __DMA2D_HandleTypeDef::XferErrorCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D transfer error callback.
- ***void(* __DMA2D_HandleTypeDef::LineEventCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D line event callback.
- ***void(* __DMA2D_HandleTypeDef::CLUTLoadingCpltCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D CLUT loading completion callback
- ***void(* __DMA2D_HandleTypeDef::MspInitCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D Msp Init callback.
- ***void(* __DMA2D_HandleTypeDef::MspDeInitCallback)(struct __DMA2D_HandleTypeDef *hdma2d)***
  DMA2D Msp DeInit callback.
- ***DMA2D_LayerCfgTypeDef __DMA2D_HandleTypeDef::LayerCfg[MAX_DMA2D_LAYER]***
  DMA2D Layers parameters
- ***HAL_LockTypeDef __DMA2D_HandleTypeDef::Lock***
  DMA2D lock.
- ***__IO HAL_DMA2D_StateTypeDef __DMA2D_HandleTypeDef::State***
  DMA2D transfer state.
- ***__IO uint32_t __DMA2D_HandleTypeDef::ErrorCode***
  DMA2D error code.

20.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

20.2.1 How to use this driver

1. Program the required configuration through the following parameters: the transfer mode, the output color mode and the output offset using HAL_DMA2D_Init() function.
2. Program the required configuration through the following parameters: the input color mode, the input color, the input alpha value, the alpha mode, the red/blue swap mode, the inverted alpha mode and the input offset using HAL_DMA2D_ConfigLayer() function for foreground or/and background layer.

Polling mode IO operation

1. Configure pdata parameter (explained hereafter), destination and data length and enable the transfer using HAL_DMA2D_Start().
2. Wait for end of transfer using HAL_DMA2D_PollForTransfer(), at this stage user can specify the value of timeout according to his end application.

Interrupt mode IO operation

1. Configure pdata parameter, destination and data length and enable the transfer using HAL_DMA2D_Start_IT().
2. Use HAL_DMA2D_IRQHandler() called under DMA2D_IRQHandler() interrupt subroutine.
3. At the end of data transfer HAL_DMA2D_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback (member of DMA2D handle structure).

4. In case of error, the HAL_DMA2D_IRQHandler() function calls the callback XferErrorCallback.

Note: In Register-to-Memory transfer mode, pdata parameter is the register color, in Memory-to-memory or Memory-to-Memory with pixel format conversion pdata is the source address.

Note: Configure the foreground source address, the background source address, the destination and data length then Enable the transfer using HAL_DMA2D_BlendingStart() in polling mode and HAL_DMA2D_BlendingStart_IT() in interrupt mode.

Note: HAL_DMA2D_BlendingStart() and HAL_DMA2D_BlendingStart_IT() functions are used if the memory to memory with blending transfer mode is selected.

5. Optionally, configure and enable the CLUT using HAL_DMA2D_CLUTLoad() in polling mode or HAL_DMA2D_CLUTLoad_IT() in interrupt mode.
6. Optionally, configure the line watermark in using the API HAL_DMA2D_ProgramLineEvent().
7. Optionally, configure the dead time value in the AHB clock cycle inserted between two consecutive accesses on the AHB master port in using the API HAL_DMA2D_ConfigDeadTime() and enable/disable the functionality with the APIs HAL_DMA2D_EnableDeadTime() or HAL_DMA2D_DisableDeadTime().
8. The transfer can be suspended, resumed and aborted using the following functions: HAL_DMA2D_Suspend(), HAL_DMA2D_Resume(), HAL_DMA2D_Abort().
9. The CLUT loading can be suspended, resumed and aborted using the following functions: HAL_DMA2D_CLUTLoading_Suspend(), HAL_DMA2D_CLUTLoading_Resume(), HAL_DMA2D_CLUTLoading_Abort().
10. To control the DMA2D state, use the following function: HAL_DMA2D_GetState().
11. To read the DMA2D error code, use the following function: HAL_DMA2D_GetError().

DMA2D HAL driver macros list

Below the list of most used macros in DMA2D HAL driver :

- __HAL_DMA2D_ENABLE: Enable the DMA2D peripheral.
- __HAL_DMA2D_GET_FLAG: Get the DMA2D pending flags.
- __HAL_DMA2D_CLEAR_FLAG: Clear the DMA2D pending flags.
- __HAL_DMA2D_ENABLE_IT: Enable the specified DMA2D interrupts.
- __HAL_DMA2D_DISABLE_IT: Disable the specified DMA2D interrupts.
- __HAL_DMA2D_GET_IT_SOURCE: Check whether the specified DMA2D interrupt is enabled or not.

Callback registration

1. The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL_DMA2D_RegisterCallback() to register a user callback.
2. Function @ref HAL_DMA2D_RegisterCallback() allows to register following callbacks: (+) XferCpltCallback : callback for transfer complete. (+) XferErrorCallback : callback for transfer error. (+) LineEventCallback : callback for line event. (+) CLUTLoadingCpltCallback : callback for CLUT loading completion. (+) MspInitCallback : DMA2D MspInit. (+) MspDeInitCallback : DMA2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function @ref HAL_DMA2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_DMA2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) XferCpltCallback : callback for transfer complete. (+) XferErrorCallback : callback for transfer error. (+) LineEventCallback : callback for line event. (+) CLUTLoadingCpltCallback : callback for CLUT loading completion. (+) MspInitCallback : DMA2D MspInit. (+) MspDeInitCallback : DMA2D MspDeInit.

4. By default, after the @ref HAL_DMA2D_Init and if the state is HAL_DMA2D_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL_DMA2D_LineEventCallback(), @ref HAL_DMA2D_CLUTLoadingCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_DMA2D_RegisterCallback before calling @ref HAL_DMA2D_DeInit or @ref HAL_DMA2D_Init function. When The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

(#) The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL_DMA2D_RegisterCallback() to register a user callback. (#) Function @ref HAL_DMA2D_RegisterCallback() allows to register following callbacks:

- XferCpltCallback : callback for transfer complete.
- XferErrorCallback : callback for transfer error.
- LineEventCallback : callback for line event.
- CLUTLoadingCpltCallback : callback for CLUT loading completion.
- MspInitCallback : DMA2D MspInit.
- MspDeInitCallback : DMA2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function @ref HAL_DMA2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_DMA2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- XferCpltCallback : callback for transfer complete.
- XferErrorCallback : callback for transfer error.
- LineEventCallback : callback for line event.
- CLUTLoadingCpltCallback : callback for CLUT loading completion.
- MspInitCallback : DMA2D MspInit.
- MspDeInitCallback : DMA2D MspDeInit. (#) By default, after the @ref HAL_DMA2D_Init and if the state is HAL_DMA2D_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples @ref HAL_DMA2D_LineEventCallback(), @ref HAL_DMA2D_CLUTLoadingCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the @ref HAL_DMA2D_Init and @ref HAL_DMA2D_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Exception as well for Transfer Completion and Transfer Error callbacks that are not defined as weak (surcharged) functions. They must be defined by the user to be resorted to. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_DMA2D_RegisterCallback before calling @ref HAL_DMA2D_DeInit or @ref HAL_DMA2D_Init function. When The compilation define USE_HAL_DMA2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Note: *You can refer to the DMA2D HAL driver header file for more useful macros*

20.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DMA2D
- De-initialize the DMA2D

This section contains the following APIs:

- *HAL_DMA2D_Init()*
- *HAL_DMA2D_DeInit()*
- *HAL_DMA2D_MspInit()*
- *HAL_DMA2D_MspDeInit()*
- *HAL_DMA2D_RegisterCallback()*
- *HAL_DMA2D_UnRegisterCallback()*

20.2.3 IO operation functions

This section provides functions allowing to:

- Configure the pdata, destination address and data size then start the DMA2D transfer.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer.
- Configure the pdata, destination address and data size then start the DMA2D transfer with interrupt.
- Configure the source for foreground and background, destination address and data size then start a MultiBuffer DMA2D transfer with interrupt.
- Abort DMA2D transfer.
- Suspend DMA2D transfer.
- Resume DMA2D transfer.
- Enable CLUT transfer.
- Configure CLUT loading then start transfer in polling mode.
- Configure CLUT loading then start transfer in interrupt mode.
- Abort DMA2D CLUT loading.
- Suspend DMA2D CLUT loading.
- Resume DMA2D CLUT loading.
- Poll for transfer complete.
- handle DMA2D interrupt request.
- Transfer watermark callback.
- CLUT Transfer Complete callback.

This section contains the following APIs:

- *HAL_DMA2D_Start()*
- *HAL_DMA2D_Start_IT()*
- *HAL_DMA2D_BlendingStart()*
- *HAL_DMA2D_BlendingStart_IT()*
- *HAL_DMA2D_Abort()*
- *HAL_DMA2D_Suspend()*
- *HAL_DMA2D_Resume()*
- *HAL_DMA2D_EnableCLUT()*
- *HAL_DMA2D_CLUTStartLoad()*
- *HAL_DMA2D_CLUTStartLoad_IT()*
- *HAL_DMA2D_CLUTLoad()*
- *HAL_DMA2D_CLUTLoad_IT()*
- *HAL_DMA2D_CLUTLoading_Abort()*
- *HAL_DMA2D_CLUTLoading_Suspend()*
- *HAL_DMA2D_CLUTLoading_Resume()*
- *HAL_DMA2D_PollForTransfer()*
- *HAL_DMA2D_IRQHandler()*
- *HAL_DMA2D_LineEventCallback()*
- *HAL_DMA2D_CLUTLoadingCpltCallback()*

20.2.4 Peripheral Control functions

This section provides functions allowing to:
- Configure the DMA2D foreground or background layer parameters.
- Configure the DMA2D CLUT transfer.
- Configure the line watermark
- Configure the dead time value.
- Enable or disable the dead time value functionality.

This section contains the following APIs:
- *HAL_DMA2D_ConfigLayer()*
- *HAL_DMA2D_ConfigCLUT()*
- *HAL_DMA2D_ProgramLineEvent()*
- *HAL_DMA2D_EnableDeadTime()*
- *HAL_DMA2D_DisableDeadTime()*
- *HAL_DMA2D_ConfigDeadTime()*

20.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to:
- Get the DMA2D state
- Get the DMA2D error code

This section contains the following APIs:
- *HAL_DMA2D_GetState()*
- *HAL_DMA2D_GetError()*

20.2.6 Detailed description of functions

HAL_DMA2D_Init

Function name

HAL_StatusTypeDef HAL_DMA2D_Init (DMA2D_HandleTypeDef * hdma2d)

Function description

Initialize the DMA2D according to the specified parameters in the DMA2D_InitTypeDef and create the associated handle.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_DeInit

Function name

HAL_StatusTypeDef HAL_DMA2D_DeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

Deinitializes the DMA2D peripheral registers to their default reset values.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_MspInit

Function name

void HAL_DMA2D_MspInit (DMA2D_HandleTypeDef * hdma2d)

Function description

Initializes the DMA2D MSP.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_MspDeInit

Function name

void HAL_DMA2D_MspDeInit (DMA2D_HandleTypeDef * hdma2d)

Function description

DeInitializes the DMA2D MSP.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA2D_RegisterCallback (DMA2D_HandleTypeDef * hdma2d, HAL_DMA2D_CallbackIDTypeDef CallbackID, pDMA2D_CallbackTypeDef pCallback)

Function description

Register a User DMA2D Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hdma2d:** DMA2D handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_DMA2D_TRANSFERCOMPLETE_CB_ID DMA2D transfer complete Callback ID
  – HAL_DMA2D_TRANSFERERROR_CB_ID DMA2D transfer error Callback ID
  – HAL_DMA2D_LINEEVENT_CB_ID DMA2D line event Callback ID
  – HAL_DMA2D_CLUTLOADINGCPLT_CB_ID DMA2D CLUT loading completion Callback ID
  – HAL_DMA2D_MSPINIT_CB_ID DMA2D MspInit callback ID
  – HAL_DMA2D_MSPDEINIT_CB_ID DMA2D MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

Notes

- No weak predefined callbacks are defined for HAL_DMA2D_TRANSFERCOMPLETE_CB_ID or HAL_DMA2D_TRANSFERERROR_CB_ID

HAL_DMA2D_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA2D_UnRegisterCallback (DMA2D_HandleTypeDef * hdma2d, HAL_DMA2D_CallbackIDTypeDef CallbackID)

Function description

Unregister a DMA2D Callback DMA2D Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hdma2d:** DMA2D handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_DMA2D_TRANSFERCOMPLETE_CB_ID DMA2D transfer complete Callback ID
  – HAL_DMA2D_TRANSFERERROR_CB_ID DMA2D transfer error Callback ID
  – HAL_DMA2D_LINEEVENT_CB_ID DMA2D line event Callback ID
  – HAL_DMA2D_CLUTLOADINGCPLT_CB_ID DMA2D CLUT loading completion Callback ID
  – HAL_DMA2D_MSPINIT_CB_ID DMA2D MspInit callback ID
  – HAL_DMA2D_MSPDEINIT_CB_ID DMA2D MspDeInit callback ID

Return values

- **status:**

Notes

- No weak predefined callbacks are defined for HAL_DMA2D_TRANSFERCOMPLETE_CB_ID or HAL_DMA2D_TRANSFERERROR_CB_ID

HAL_DMA2D_Start

Function name

HAL_StatusTypeDef HAL_DMA2D_Start (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the DMA2D Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_BlendingStart

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the multi-source DMA2D Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address for the foreground layer.
- **SrcAddress2:** The source memory Buffer address for the background layer.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_Start_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t pdata, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **pdata:** Configure the source memory Buffer address if the Memory-to-Memory or Memory-to-Memory with pixel format conversion mode is selected, or configure the color value if Register-to-Memory mode is selected.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_BlendingStart_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_BlendingStart_IT (DMA2D_HandleTypeDef * hdma2d, uint32_t SrcAddress1, uint32_t SrcAddress2, uint32_t DstAddress, uint32_t Width, uint32_t Height)

Function description

Start the multi-source DMA2D Transfer with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **SrcAddress1:** The source memory Buffer address for the foreground layer.
- **SrcAddress2:** The source memory Buffer address for the background layer.
- **DstAddress:** The destination memory Buffer address.
- **Width:** The width of data to be transferred from source to destination (expressed in number of pixels per line).
- **Height:** The height of data to be transferred from source to destination (expressed in number of lines).

Return values

- **HAL:** status

HAL_DMA2D_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_Suspend (DMA2D_HandleTypeDef * hdma2d)

Function description

Suspend the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_Resume (DMA2D_HandleTypeDef * hdma2d)

Function description

Resume the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_Abort (DMA2D_HandleTypeDef * hdma2d)

Function description

Abort the DMA2D Transfer.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_EnableCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableCLUT (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Enable the DMA2D CLUT Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTStartLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTStartLoad (DMA2D_HandleTypeDef * hdma2d, const DMA2D_CLUTCfgTypeDef * CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTStartLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTStartLoad_IT (DMA2D_HandleTypeDef * hdma2d, const DMA2D_CLUTCfgTypeDef * CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoad

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL_DMA2D_CLUTStartLoad() instead to benefit from code compactness, code size and improved heap usage.

HAL_DMA2D_CLUTLoad_IT

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoad_IT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Start DMA2D CLUT Loading with interrupt enabled.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL_DMA2D_CLUTStartLoad_IT() instead to benefit from code compactness, code size and improved heap usage.

HAL_DMA2D_CLUTLoading_Abort

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Abort (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Abort the DMA2D CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Suspend

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Suspend (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Suspend the DMA2D CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_CLUTLoading_Resume

Function name

HAL_StatusTypeDef HAL_DMA2D_CLUTLoading_Resume (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Resume the DMA2D CLUT loading.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA2D_PollForTransfer (DMA2D_HandleTypeDef * hdma2d, uint32_t Timeout)

Function description

Polling for transfer complete or CLUT loading.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_DMA2D_IRQHandler

Function name

void HAL_DMA2D_IRQHandler (DMA2D_HandleTypeDef * hdma2d)

Function description

Handle DMA2D interrupt request.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** status

HAL_DMA2D_LineEventCallback

Function name

void HAL_DMA2D_LineEventCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

Transfer watermark callback.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_CLUTLoadingCpltCallback

Function name

void HAL_DMA2D_CLUTLoadingCpltCallback (DMA2D_HandleTypeDef * hdma2d)

Function description

CLUT Transfer Complete callback.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **None:**

HAL_DMA2D_ConfigLayer

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigLayer (DMA2D_HandleTypeDef * hdma2d, uint32_t LayerIdx)

Function description

Configure the DMA2D Layer according to the specified parameters in the DMA2D_HandleTypeDef.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

HAL_DMA2D_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigCLUT (DMA2D_HandleTypeDef * hdma2d, DMA2D_CLUTCfgTypeDef CLUTCfg, uint32_t LayerIdx)

Function description

Configure the DMA2D CLUT Transfer.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **CLUTCfg:** Pointer to a DMA2D_CLUTCfgTypeDef structure that contains the configuration information for the color look up table.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Return values

- **HAL:** status

Notes

- API obsolete and maintained for compatibility with legacy. User is invited to resort to HAL_DMA2D_CLUTStartLoad() instead to benefit from code compactness, code size and improved heap usage.

HAL_DMA2D_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_DMA2D_ProgramLineEvent (DMA2D_HandleTypeDef * hdma2d, uint32_t Line)

Function description

Configure the line watermark.

Parameters

- **hdma2d:** Pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.
- **Line:** Line Watermark configuration (maximum 16-bit long value expected).

Return values

- **HAL:** status

Notes

- HAL_DMA2D_ProgramLineEvent() API enables the transfer watermark interrupt.
- The transfer watermark interrupt is disabled once it has occurred.

HAL_DMA2D_EnableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_EnableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

Enable DMA2D dead time feature.

Parameters

- **hdma2d:** DMA2D handle.

Return values

- **HAL:** status

HAL_DMA2D_DisableDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_DisableDeadTime (DMA2D_HandleTypeDef * hdma2d)

Function description

Disable DMA2D dead time feature.

Parameters

- **hdma2d:** DMA2D handle.

Return values

- **HAL:** status

HAL_DMA2D_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_DMA2D_ConfigDeadTime (DMA2D_HandleTypeDef * hdma2d, uint8_t DeadTime)

Function description

Configure dead time.

Parameters

- **hdma2d:** DMA2D handle.
- **DeadTime:** dead time value.

Return values

- **HAL:** status

Notes

- The dead time value represents the guaranteed minimum number of cycles between two consecutive transactions on the AHB bus.

HAL_DMA2D_GetState

Function name

HAL_DMA2D_StateTypeDef HAL_DMA2D_GetState (const DMA2D_HandleTypeDef * hdma2d)

Function description

Return the DMA2D state.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for the DMA2D.

Return values

- **HAL:** state

HAL_DMA2D_GetError

Function name

uint32_t HAL_DMA2D_GetError (const DMA2D_HandleTypeDef * hdma2d)

Function description

Return the DMA2D error code.

Parameters

- **hdma2d:** pointer to a DMA2D_HandleTypeDef structure that contains the configuration information for DMA2D.

Return values

- **DMA2D:** Error Code

20.3 DMA2D Firmware driver defines

The following section lists the various define and macros of the module.

20.3.1 DMA2D

DMA2D

DMA2D Alpha Inversion

DMA2D_REGULAR_ALPHA

No modification of the alpha channel value

DMA2D_INVERTED_ALPHA

Invert the alpha channel value

DMA2D Alpha Mode

DMA2D_NO_MODIF_ALPHA

No modification of the alpha channel value

DMA2D_REPLACE_ALPHA

Replace original alpha channel value by programmed alpha value

DMA2D_COMBINE_ALPHA

Replace original alpha channel value by programmed alpha value with original alpha channel value

DMA2D Bytes Swap

DMA2D_BYTES_REGULAR

Bytes in regular order in output FIFO

DMA2D_BYTES_SWAP

Bytes are swapped two by two in output FIFO

DMA2D Chroma Sub Sampling

DMA2D_NO_CSS

No chroma sub-sampling 4:4:4

DMA2D_CSS_422

chroma sub-sampling 4:2:2

DMA2D_CSS_420

chroma sub-sampling 4:2:0

DMA2D CLUT Color Mode

DMA2D_CCM_ARGB8888

ARGB8888 DMA2D CLUT color mode

DMA2D_CCM_RGB888

RGB888 DMA2D CLUT color mode

DMA2D CLUT Size

DMA2D_CLUT_SIZE

DMA2D maximum CLUT size

DMA2D Color Value

DMA2D_COLOR_VALUE

Color value mask

DMA2D Error Code

HAL_DMA2D_ERROR_NONE

No error

HAL_DMA2D_ERROR_TE

Transfer error

HAL_DMA2D_ERROR_CE

Configuration error

HAL_DMA2D_ERROR_CAE

CLUT access error

HAL_DMA2D_ERROR_TIMEOUT

Timeout error

HAL_DMA2D_ERROR_INVALID_CALLBACK

Invalid callback error

DMA2D Exported Macros

__HAL_DMA2D_RESET_HANDLE_STATE

Description:

- Reset DMA2D handle state.

Parameters:

- __HANDLE__: specifies the DMA2D handle.

Return value:

- None

__HAL_DMA2D_ENABLE

Description:

- Enable the DMA2D.

Parameters:

- __HANDLE__: DMA2D handle

Return value:

- None.

__HAL_DMA2D_GET_FLAG

Description:

- Get the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  – DMA2D_FLAG_CE: Configuration error flag
  – DMA2D_FLAG_CTC: CLUT transfer complete flag
  – DMA2D_FLAG_CAE: CLUT access error flag
  – DMA2D_FLAG_TW: Transfer Watermark flag
  – DMA2D_FLAG_TC: Transfer complete flag
  – DMA2D_FLAG_TE: Transfer error flag

Return value:

- The: state of FLAG.

__HAL_DMA2D_CLEAR_FLAG

Description:

- Clear the DMA2D pending flags.

Parameters:

- __HANDLE__: DMA2D handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  – DMA2D_FLAG_CE: Configuration error flag
  – DMA2D_FLAG_CTC: CLUT transfer complete flag
  – DMA2D_FLAG_CAE: CLUT access error flag
  – DMA2D_FLAG_TW: Transfer Watermark flag
  – DMA2D_FLAG_TC: Transfer complete flag
  – DMA2D_FLAG_TE: Transfer error flag

Return value:

- None

__HAL_DMA2D_ENABLE_IT

Description:
- Enable the specified DMA2D interrupts.

Parameters:
- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be enabled. This parameter can be any combination of the following values:
  – DMA2D_IT_CE: Configuration error interrupt mask
  – DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  – DMA2D_IT_CAE: CLUT access error interrupt mask
  – DMA2D_IT_TW: Transfer Watermark interrupt mask
  – DMA2D_IT_TC: Transfer complete interrupt mask
  – DMA2D_IT_TE: Transfer error interrupt mask

Return value:
- None

__HAL_DMA2D_DISABLE_IT

Description:
- Disable the specified DMA2D interrupts.

Parameters:
- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt sources to be disabled. This parameter can be any combination of the following values:
  – DMA2D_IT_CE: Configuration error interrupt mask
  – DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  – DMA2D_IT_CAE: CLUT access error interrupt mask
  – DMA2D_IT_TW: Transfer Watermark interrupt mask
  – DMA2D_IT_TC: Transfer complete interrupt mask
  – DMA2D_IT_TE: Transfer error interrupt mask

Return value:
- None

__HAL_DMA2D_GET_IT_SOURCE

Description:
- Check whether the specified DMA2D interrupt source is enabled or not.

Parameters:
- __HANDLE__: DMA2D handle
- __INTERRUPT__: specifies the DMA2D interrupt source to check. This parameter can be one of the following values:
  – DMA2D_IT_CE: Configuration error interrupt mask
  – DMA2D_IT_CTC: CLUT transfer complete interrupt mask
  – DMA2D_IT_CAE: CLUT access error interrupt mask
  – DMA2D_IT_TW: Transfer Watermark interrupt mask
  – DMA2D_IT_TC: Transfer complete interrupt mask
  – DMA2D_IT_TE: Transfer error interrupt mask

Return value:
- The: state of INTERRUPT source.

DMA2D Exported Types

MAX_DMA2D_LAYER

DMA2D maximum number of layers

DMA2D Flags

DMA2D_FLAG_CE

Configuration Error Interrupt Flag

DMA2D_FLAG_CTC

CLUT Transfer Complete Interrupt Flag

DMA2D_FLAG_CAE

CLUT Access Error Interrupt Flag

DMA2D_FLAG_TW

Transfer Watermark Interrupt Flag

DMA2D_FLAG_TC

Transfer Complete Interrupt Flag

DMA2D_FLAG_TE

Transfer Error Interrupt Flag

DMA2D Input Color Mode

DMA2D_INPUT_ARGB8888

ARGB8888 color mode

DMA2D_INPUT_RGB888

RGB888 color mode

DMA2D_INPUT_RGB565

RGB565 color mode

DMA2D_INPUT_ARGB1555

ARGB1555 color mode

DMA2D_INPUT_ARGB4444

ARGB4444 color mode

DMA2D_INPUT_L8

L8 color mode

DMA2D_INPUT_AL44

AL44 color mode

DMA2D_INPUT_AL88

AL88 color mode

DMA2D_INPUT_L4

L4 color mode

DMA2D_INPUT_A8

A8 color mode

DMA2D_INPUT_A4

A4 color mode

DMA2D_INPUT_YCBCR

YCbCr color mode

DMA2D Interrupts

DMA2D_IT_CE

Configuration Error Interrupt

DMA2D_IT_CTC

CLUT Transfer Complete Interrupt

DMA2D_IT_CAE

CLUT Access Error Interrupt

DMA2D_IT_TW

Transfer Watermark Interrupt

DMA2D_IT_TC

Transfer Complete Interrupt

DMA2D_IT_TE

Transfer Error Interrupt

DMA2D Layers

DMA2D_BACKGROUND_LAYER

DMA2D Background Layer (layer 0)

DMA2D_FOREGROUND_LAYER

DMA2D Foreground Layer (layer 1)

DMA2D Line Offset Mode

DMA2D_LOM_PIXELS

Line offsets expressed in pixels

DMA2D_LOM_BYTES

Line offsets expressed in bytes

DMA2D Maximum Line Watermark

DMA2D_LINE_WATERMARK_MAX

DMA2D maximum line watermark

DMA2D Maximum Number of Layers

DMA2D_MAX_LAYER

DMA2D maximum number of layers

DMA2D Mode

DMA2D_M2M

DMA2D memory to memory transfer mode

DMA2D_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

DMA2D_M2M_BLEND

DMA2D memory to memory with blending transfer mode

DMA2D_R2M

DMA2D register to memory transfer mode

DMA2D_M2M_BLEND_FG

DMA2D memory to memory with blending transfer mode and fixed color FG

DMA2D_M2M_BLEND_BG

DMA2D memory to memory with blending transfer mode and fixed color BG

DMA2D Offset

DMA2D_OFFSET

maximum Line Offset

DMA2D Output Color Mode

DMA2D_OUTPUT_ARGB8888

ARGB8888 DMA2D color mode

DMA2D_OUTPUT_RGB888

RGB888 DMA2D color mode

DMA2D_OUTPUT_RGB565

RGB565 DMA2D color mode

DMA2D_OUTPUT_ARGB1555

ARGB1555 DMA2D color mode

DMA2D_OUTPUT_ARGB4444

ARGB4444 DMA2D color mode

DMA2D Red and Blue Swap

DMA2D_RB_REGULAR

Select regular mode (RGB or ARGB)

DMA2D_RB_SWAP

Select swap mode (BGR or ABGR)

DMA2D Size

DMA2D_PIXEL

DMA2D maximum number of pixels per line

DMA2D_LINE

DMA2D maximum number of lines

DMA2D Time Out

DMA2D_TIMEOUT_ABORT

1s

DMA2D_TIMEOUT_SUSPEND

1s

21 HAL DMA Generic Driver

21.1 DMA Firmware driver registers structures

21.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32u5xx_hal_dma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t BlkHWRequest***
- ***uint32_t Direction***
- ***uint32_t SrcInc***
- ***uint32_t DestInc***
- ***uint32_t SrcDataWidth***
- ***uint32_t DestDataWidth***
- ***uint32_t Priority***
- ***uint32_t SrcBurstLength***
- ***uint32_t DestBurstLength***
- ***uint32_t TransferAllocatedPort***
- ***uint32_t TransferEventMode***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
  Specifies the DMA channel request. This parameter can be a value of ***DMA_Request_Selection***
- ***uint32_t DMA_InitTypeDef::BlkHWRequest***
  Specifies the Block hardware request mode for DMA channel. Block Hardware request feature can be used only with dedicated peripherals. This parameter can be a value of ***DMA_Block_Request***
- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies the transfer direction for DMA channel. This parameter can be a value of ***DMA_Transfer_Direction***
- ***uint32_t DMA_InitTypeDef::SrcInc***
  Specifies the source increment mode for the DMA channel. This parameter can be a value of ***DMA_Source_Increment_Mode***
- ***uint32_t DMA_InitTypeDef::DestInc***
  Specifies the destination increment mode for the DMA channel. This parameter can be a value of ***DMA_Destination_Increment_Mode***
- ***uint32_t DMA_InitTypeDef::SrcDataWidth***
  Specifies the source data width for the DMA channel. This parameter can be a value of ***DMA_Source_Data_Width***
- ***uint32_t DMA_InitTypeDef::DestDataWidth***
  Specifies the destination data width for the DMA channel. This parameter can be a value of ***DMA_Destination_Data_Width***
- ***uint32_t DMA_InitTypeDef::Priority***
  Specifies the priority level for the DMA channel. This parameter can be a value of ***DMA_Priority_Level***
- ***uint32_t DMA_InitTypeDef::SrcBurstLength***
  Specifies the source burst length (number of beats within a burst) for the DMA channel. This parameter can be a value between 1 and 64
- ***uint32_t DMA_InitTypeDef::DestBurstLength***
  Specifies the destination burst length (number of beats within a burst) for the DMA channel. This parameter can be a value between 1 and 64
- ***uint32_t DMA_InitTypeDef::TransferAllocatedPort***
  Specifies the transfer allocated ports. This parameter can be a combination of ***DMA_Transfer_Allocated_Port***

- ***uint32_t DMA_InitTypeDef::TransferEventMode***
  Specifies the transfer event mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Event_Mode***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the transfer mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Mode***

21.1.2 DMA_InitLinkedListTypeDef

DMA_InitLinkedListTypeDef is defined in the stm32u5xx_hal_dma.h

Data Fields

- ***uint32_t Priority***
- ***uint32_t LinkStepMode***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t TransferEventMode***
- ***uint32_t LinkedListMode***

Field Documentation

- ***uint32_t DMA_InitLinkedListTypeDef::Priority***
  Specifies the priority level for the DMA channel. This parameter can be a value of ***DMA_Priority_Level***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkStepMode***
  Specifies the link step mode for the DMA channel. This parameter can be a value of ***DMAEx_Link_Step_Mode***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkAllocatedPort***
  Specifies the linked-list allocated port for the DMA channel. This parameter can be a value of ***DMAEx_Link_Allocated_Port***
- ***uint32_t DMA_InitLinkedListTypeDef::TransferEventMode***
  Specifies the transfer event mode for the DMA channel. This parameter can be a value of ***DMA_Transfer_Event_Mode***
- ***uint32_t DMA_InitLinkedListTypeDef::LinkedListMode***
  Specifies linked-list transfer mode for the DMA channel. This parameter can be a value of ***DMAEx_LinkedList_Mode***

21.1.3 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32u5xx_hal_dma.h

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***DMA_InitLinkedListTypeDef InitLinkedList***
- ***HAL_LockTypeDef Lock***
- ***uint32_t Mode***
- ***__IO HAL_DMA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***void(* XferAbortCallback***
- ***void(* XferSuspendCallback***
- ***struct __DMA_QListTypeDef * LinkedListQueue***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register the DMA channel base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA channel init parameters

- ***DMA_InitLinkedListTypeDef __DMA_HandleTypeDef::InitLinkedList***
  DMA channel linked-list init parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***uint32_t __DMA_HandleTypeDef::Mode***
  DMA transfer mode
- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA error code
- ***void* __DMA_HandleTypeDef::Parent***
  Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer error callback
- ***void(* __DMA_HandleTypeDef::XferAbortCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Abort callback
- ***void(* __DMA_HandleTypeDef::XferSuspendCallback)(struct __DMA_HandleTypeDef *hdma)***
  DMA transfer Suspend callback
- ***struct __DMA_QListTypeDef* __DMA_HandleTypeDef::LinkedListQueue***
  DMA linked-list queue

21.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

21.2.1 How to use this driver

DMA transfer modes are divided to 2 major categories :

- Normal transfers (legacy)
- Linked-list transfers

Normal transfers mode is initialized via the standard module and linked-list mode is configured via the extended module.

Additionally to linked-list capability, all advanced DMA features are managed and configured via the extended module as extensions to normal mode. Advanced features are :

- Repeated block feature.
- Trigger feature.
- Data handling feature.

DMA Legacy circular transfer, is replaced by circular linked-list configuration.

Initialization and De-Initialization

For a given channel, enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM/FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests.

For a given channel, use HAL_DMA_Init function to program the required configuration for normal transfer through the following parameters:

- Request : Specifies the DMA channel request Request parameters :
  – can be a value of DMA_Request_Selection
- BlkHWRequest : Specifies the Block hardware request mode for DMA channel
  – can be a value of DMA_Block_Request
- Direction : Specifies the transfer direction for DMA channel
  – can be a value of DMA_Transfer_Direction

- SrcInc : Specifies the source increment mode for the DMA channel
  – can be a value of DMA_Source_Increment_Mode
- DestInc : Specifies the destination increment mode for the DMA channel
  – can be a value of DMA_Destination_Increment_Mode
- SrcDataWidth : Specifies the source data width for the DMA channel
  – can be a value of DMA_Source_Data_Width
- DestDataWidth : Specifies the destination data width for the DMA channel
  – can be a value of DMA_Destination_Data_Width
- Priority : Specifies the priority for the DMA channel
  – can be a value of DMA_Priority_Level
- SrcBurstLength : Specifies the source burst length (number of beats) for the DMA channel
  – can be a value of between 1 and 64
- DestBurstLength : Specifies the destination burst length (number of beats) for the DMA channel
  – can be a value of between 1 and 64
- TransferAllocatedPort : Specifies the source and destination allocated ports
  – can be a value of DMA_Transfer_Allocated_Port
- TransferEventMode : Specifies the transfer event mode for the DMA channel
  – can be a value of DMA_Transfer_Event_Mode
- Mode : Specifies the transfer mode for the DMA channel
  – can be DMA_NORMAL

Polling mode IO operation

- Use HAL_DMA_Start() to start a DMA normal transfer after the configuration of source address, destination address and the size of data to be transferred.
- Use HAL_DMA_PollForTransfer() to poll for selected transfer level. In this case a fixed Timeout can be configured by User depending on his application. Transfer level can be :
  – HAL_DMA_HALF_TRANSFER
  – HAL_DMA_FULL_TRANSFER For circular transfer, this API returns an HAL_ERROR with HAL_DMA_ERROR_NOT_SUPPORTED error code.
- Use HAL_DMA_Abort() function to abort any ongoing DMA transfer in blocking mode. This API returns HAL_ERROR when there is no ongoing transfer or timeout is reached when disabling the DMA channel. (This API should not be called from an interrupt service routine)

Interrupt mode IO operation

- Configure the DMA interrupt priority using HAL_NVIC_SetPriority()
- Enable the DMA IRQ handler using HAL_NVIC_EnableIRQ()
- Use HAL_DMA_RegisterCallback() function to register user callbacks from the following list :
  – XferCpltCallback : transfer complete callback.
  – XferHalfCpltCallback : half transfer complete callback.
  – XferErrorCallback : transfer error callback.
  – XferAbortCallback : transfer abort complete callback.
  – XferSuspendCallback : transfer suspend complete callback.
- Use HAL_DMA_Start_IT() to start the DMA transfer after the enable of DMA interrupts and the configuration of source address,destination address and the size of data to be transferred.
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() interrupt subroutine to handle any DMA interrupt.
- Use HAL_DMA_Abort_IT() function to abort any on-going DMA transfer in non-blocking mode. This API will suspend immediately the DMA channel execution. When the transfer is effectively suspended, an interrupt is generated and HAL_DMA_IRQHandler() will reset the channel and execute the callback XferAbortCallback. (This API could be called from an interrupt service routine)

State and errors

- Use HAL_DMA_GetState() function to get the DMA state.
- Use HAL_DMA_GetError() function to get the DMA error code.

Security and privilege attributes

- Use HAL_DMA_ConfigChannelAttributes() function to configure DMA channel security and privilege attributes.
  – Security : at channel level, at source level and at destination level.
  – Privilege : at channel level.
- Use HAL_DMA_GetConfigChannelAttributes() function to get the DMA channel attributes.
- Use HAL_DMA_LockChannelAttributes() function to lock the DMA channel security and privilege attributes configuration. This API can be called once after each system boot. If called again, HAL_DMA_ConfigChannelAttributes() API has no effect. Unlock is done either by a system boot or a by an RCC reset.
- Use HAL_DMA_GetLockChannelAttributes() function to get the attributes lock status.

DMA HAL driver macros list

Below the list of most used macros in DMA HAL driver.

- __HAL_DMA_ENABLE : Enable the specified DMA Channel.
- __HAL_DMA_DISABLE : Disable the specified DMA Channel.
- __HAL_DMA_GET_FLAG : Get the DMA Channel pending flags.
- __HAL_DMA_CLEAR_FLAG : Clear the DMA Channel pending flags.
- __HAL_DMA_ENABLE_IT : Enable the specified DMA Channel interrupts.
- __HAL_DMA_DISABLE_IT : Disable the specified DMA Channel interrupts.
- __HAL_DMA_GET_IT_SOURCE : Check whether the specified DMA Channel interrupt has occurred or not.

Note: You can refer to the header file of the DMA HAL driver for more useful macros.

21.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize and de-initialize the DMA channel in normal mode.

- The HAL_DMA_Init() function follows the DMA channel configuration procedures as described in reference manual.
- The HAL_DMA_DeInit() function allows to de-initialize the DMA channel.

This section contains the following APIs:

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

21.2.3 IO operation functions

This section provides functions allowing to :

- Configure the source, destination address and data size and Start DMA transfer in normal mode
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request
- Register and Unregister DMA callbacks
- The HAL_DMA_Start() function allows to start the DMA channel transfer in normal mode (Blocking mode).
- The HAL_DMA_Start_IT() function allows to start the DMA channel transfer in normal mode (Non-blocking mode).
- The HAL_DMA_Abort() function allows to abort any on-going transfer (Blocking mode).
- The HAL_DMA_Abort_IT() function allows to abort any on-going transfer (Non-blocking mode).

- The HAL_DMA_PollForTransfer() function allows to poll on half transfer and transfer complete (Blocking mode). This API cannot be used for circular transfers.
- The HAL_DMA_IRQHandler() function allows to handle any DMA channel interrupt (Non-blocking mode).
- The HAL_DMA_RegisterCallback() and HAL_DMA_UnRegisterCallback() functions allow respectively to register and unregister user customized callbacks. User callbacks are called under HAL_DMA_IRQHandler().

This section contains the following APIs:

- *HAL_DMA_Start()*
- *HAL_DMA_Start_IT()*
- *HAL_DMA_Abort()*
- *HAL_DMA_Abort_IT()*
- *HAL_DMA_PollForTransfer()*
- *HAL_DMA_IRQHandler()*
- *HAL_DMA_RegisterCallback()*
- *HAL_DMA_UnRegisterCallback()*

21.2.4 State and Errors functions

This section provides functions allowing to :

- Check the DMA state
- Get error code
- The HAL_DMA_GetState() function allows to get the DMA channel state.
- The HAL_DMA_DeInit() function allows to get the DMA channel error code.

This section contains the following APIs:

- *HAL_DMA_GetState()*
- *HAL_DMA_GetError()*

21.2.5 DMA Attributes functions

This section provides functions allowing to :

- Configure DMA channel secure and privilege attributes.
- Get DMA channel secure and privilege attributes.
- Lock DMA channel secure and privilege attributes configuration.
- Check whether DMA channel secure and privilege attributes configuration is locked or not.
- The HAL_DMA_ConfigChannelAttributes() function allows to configure DMA channel security and privilege attributes.
- The HAL_DMA_GetConfigChannelAttributes() function allows to get DMA channel security and privilege attributes configuration.
- The HAL_DMA_LockChannelAttributes() function allows to lock the DMA channel security and privilege attributes.
- The HAL_DMA_GetLockChannelAttributes() function allows to get the DMA channel security and privilege attributes lock status.

This section contains the following APIs:

- *HAL_DMA_ConfigChannelAttributes()*
- *HAL_DMA_GetConfigChannelAttributes()*
- *HAL_DMA_LockChannelAttributes()*
- *HAL_DMA_GetLockChannelAttributes()*

21.2.6 Detailed description of functions

HAL_DMA_Init

Function name

HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef *const hdma)

Function description

Initialize the DMA channel in normal mode according to the specified parameters in the DMA_InitTypeDef and create the associated handle.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_DeInit

Function name

HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef *const hdma)

Function description

DeInitialize the DMA channel when it is configured in normal mode.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_Start

Function name

HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef *const hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SrcDataSize)

Function description

Start the DMA channel transfer in normal mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** : The source data address.
- **DstAddress:** : The destination data address.
- **SrcDataSize:** : The length of data to be transferred from source to destination in bytes.

Return values

- **HAL:** status.

HAL_DMA_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef *const hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t SrcDataSize)

Function description

Starts the DMA channel transfer in normal mode with interrupts enabled (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** : The source data address.
- **DstAddress:** : The destination data address.
- **SrcDataSize:** : The length of data to be transferred from source to destination in bytes.

Return values

- **HAL:** status.

HAL_DMA_Abort

Function name

HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef *const hdma)

Function description

Abort any on-going DMA channel transfer (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

Notes

- After suspending a DMA channel, a wait until the DMA channel is effectively stopped is added. If a channel is suspended while a data transfer is on-going, the current data will be transferred and the channel will be effectively suspended only after the transfer of any on-going data is finished.

HAL_DMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef *const hdma)

Function description

Abort any on-going DMA channel transfer in interrupt mode (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef *const hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer status (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CompleteLevel:** : Specifies the DMA level complete.
- **Timeout:** : Timeout duration.

Return values

- **HAL:** status

HAL_DMA_IRQHandler

Function name

void HAL_DMA_IRQHandler (DMA_HandleTypeDef *const hdma)

Function description

Handle DMA interrupt request (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **None.:**

HAL_DMA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_RegisterCallback (DMA_HandleTypeDef *const hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *const _hdma) pCallback)

Function description

Register callback according to specified ID.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** : User Callback identifier which could be a value of HAL_DMA_CallbackIDTypeDef enumeration.
- **pCallback:** : Pointer to private callback function.

Return values

- **HAL:** status.

Notes

- The HAL_DMA_RegisterCallback() may be called before HAL_DMA_Init() in HAL_DMA_STATE_RESET to register callbacks for HAL_DMA_MSPINIT_CB_ID and HAL_DMA_MSPDEINIT_CB_ID.

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef *const hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

Unregister callback according to specified ID.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **CallbackID:** : User Callback identifier which could be a value of HAL_DMA_CallbackIDTypeDef enum.

Return values

- **HAL:** status.

Notes

- The HAL_DMA_UnRegisterCallback() may be called before HAL_DMA_Init() in HAL_DMA_STATE_RESET to un-register callbacks for HAL_DMA_MSPINIT_CB_ID and HAL_DMA_MSPDEINIT_CB_ID.

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef const *const hdma)

Function description

Returns the DMA channel state.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** state.

HAL_DMA_GetError

Function name

uint32_t HAL_DMA_GetError (DMA_HandleTypeDef const *const hdma)

Function description

Return the DMA channel error code.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **DMA:** Error Code.

HAL_DMA_ConfigChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_ConfigChannelAttributes (DMA_HandleTypeDef *const hdma, uint32_t ChannelAttributes)

Function description

Configure the DMA channel security and privilege attribute(s).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **ChannelAttributes:** : Specifies the DMA channel secure/privilege attributes. This parameter can be a one or a combination of DMA Channel Attributes.

Return values

- **HAL:** Status.

Notes

- These attributes cannot be modified when the corresponding lock state is enabled.

HAL_DMA_GetConfigChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetConfigChannelAttributes (DMA_HandleTypeDef const *const hdma, uint32_t *const pChannelAttributes)

Function description

Get the DMA channel security and privilege attributes.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pChannelAttributes:** : Pointer to the returned attributes.

Return values

- **HAL:** Status.

HAL_DMA_LockChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_LockChannelAttributes (DMA_HandleTypeDef const *const hdma)

Function description

Lock the DMA channel security and privilege attribute(s).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** Status.

HAL_DMA_GetLockChannelAttributes

Function name

HAL_StatusTypeDef HAL_DMA_GetLockChannelAttributes (DMA_HandleTypeDef const *const hdma, uint32_t *const pLockState)

Function description

Get the security and privilege attribute lock state of a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pLockState:** : Pointer to lock state (returned value can be DMA_CHANNEL_ATTRIBUTE_UNLOCKED or DMA_CHANNEL_ATTRIBUTE_LOCKED).

Return values

- **HAL:** status.

21.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

21.3.1 DMA

DMA

DMA Block Request

DMA_BREQ_SINGLE_BURST

Hardware request protocol at a single / burst level

DMA_BREQ_BLOCK

Hardware request protocol at a block level

DMA Channel Attributes

DMA_CHANNEL_PRIV

Channel is privileged

DMA_CHANNEL_NPRIV

Channel is unprivileged

DMA_CHANNEL_SEC

Channel is secure

DMA_CHANNEL_NSEC

Channel is non-secure

DMA_CHANNEL_SRC_SEC

Channel source is secure

DMA_CHANNEL_SRC_NSEC

Channel source is non-secure

DMA_CHANNEL_DEST_SEC

Channel destination is secure

DMA_CHANNEL_DEST_NSEC

Channel destination is non-secure

DMA_CHANNEL_ATTRIBUTE_UNLOCKED

Channel attribute is unlocked

DMA_CHANNEL_ATTRIBUTE_LOCKED

Channel attribute is locked

DMA destination Data Width

DMA_DEST_DATAWIDTH_BYTE

Destination data width : Byte

DMA_DEST_DATAWIDTH_HALFWORD

Destination data width : HalfWord

DMA_DEST_DATAWIDTH_WORD

Destination data width : Word

DMA Destination Increment Mode

DMA_DINC_FIXED

Destination fixed single / burst

DMA_DINC_INCREMENTED

Destination incremented single / burst

DMA Error Codes

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_DTE

Data transfer error

HAL_DMA_ERROR_ULE

Update linked-list item error

HAL_DMA_ERROR_USE

User setting error

HAL_DMA_ERROR_TO

Trigger overrun error

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_NO_XFER

No transfer ongoing error

HAL_DMA_ERROR_BUSY

Busy error

HAL_DMA_ERROR_INVALID_CALLBACK

Invalid callback error

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

DMA Exported Macros

__HAL_DMA_RESET_HANDLE_STATE

Description:

- Reset DMA handle state.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None.

__HAL_DMA_ENABLE

Description:

- Enable the specified DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None

__HAL_DMA_DISABLE

Description:

- Disable the specified DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- None

__HAL_DMA_GET_FLAG

Description:

- Get the DMA channel pending flags.

Parameters:

- __HANDLE__: : DMA handle.
- __FLAG__: : Get the specified flag. This parameter can be any combination of the following values:
  - DMA_FLAG_TC : Transfer Complete flag.
  - DMA_FLAG_HT : Half Transfer Complete flag.
  - DMA_FLAG_DTE : Data Transfer Error flag.
  - DMA_FLAG_ULE : Update linked-list Error flag.
  - DMA_FLAG_USE : User Setting Error flag.
  - DMA_FLAG_TO : Trigger Overrun flag.
  - DMA_FLAG_SUSP : Completed Suspension flag.
  - DMA_FLAG_IDLEF : Idle flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_DMA_CLEAR_FLAG

Description:

- Clear the DMA Channel pending flags.

Parameters:

- __HANDLE__: : DMA handle.
- __FLAG__: : Specifies the flag to clear. This parameter can be any combination of the following values:
  - DMA_FLAG_TC : Transfer Complete flag.
  - DMA_FLAG_HT : Half Transfer Complete flag.
  - DMA_FLAG_DTE : Data Transfer Error flag.
  - DMA_FLAG_ULE : Update Linked-List Error flag.
  - DMA_FLAG_USE : User Setting Error flag.
  - DMA_FLAG_TO : Trigger Overrun flag.
  - DMA_FLAG_SUSP : Completed Suspension flag.

Return value:

- None

__HAL_DMA_ENABLE_IT

Description:

- Enable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : Specifies the DMA interrupt sources to be enabled. This parameter can be any combination of the following values:
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- None

__HAL_DMA_DISABLE_IT

Description:

- Disable the specified DMA Channel interrupts.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : specifies the DMA interrupt sources to be disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Checks whether the specified DMA Channel interrupt is enabled or not.

Parameters:

- __HANDLE__: : DMA handle.
- __INTERRUPT__: : specifies the DMA interrupt source to check.
  - DMA_IT_TC : Transfer Complete interrupt.
  - DMA_IT_HT : Half Transfer Complete interrupt.
  - DMA_IT_DTE : Data Transfer Error interrupt.
  - DMA_IT_ULE : Update Linked-List Error interrupt.
  - DMA_IT_USE : User Setting Error interrupt.
  - DMA_IT_TO : Trigger Overrun interrupt.
  - DMA_IT_SUSP : Completed Suspension interrupt.

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_SET_COUNTER

Description:

- Writes the block number of bytes to be transferred from the source on the DMA Channel.

Parameters:

- __HANDLE__: : DMA handle.
- __COUNTER__: : Number of data bytes to be transferred from the source (from 0 to 65535).

__HAL_DMA_GET_COUNTER

Description:

- Returns the number of remaining data bytes in the current DMA Channel transfer.

Parameters:

- __HANDLE__: : DMA handle.

Return value:

- The: number of remaining data units in the current DMA Stream transfer.

DMA Flag Definition

DMA_FLAG_IDLE

Idle flag

DMA_FLAG_TC

Transfer complete flag

DMA_FLAG_HT

Half transfer complete flag

DMA_FLAG_DTE

Data transfer error flag

DMA_FLAG_ULE

Update linked-list item error flag

DMA_FLAG_USE

User setting error flag

DMA_FLAG_SUSP

Completed suspension flag

DMA_FLAG_TO

Trigger overrun flag

TIM DMA Handle Index

TIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3

Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4

Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_COMMUTATION

Index of the DMA handle used for Commutation DMA requests

TIM_DMA_ID_TRIGGER

Index of the DMA handle used for Trigger DMA requests

DMA Interrupt Enable Definition

DMA_IT_TC

Transfer complete interrupt

DMA_IT_HT

Half transfer complete interrupt

DMA_IT_DTE

Data transfer error interrupt

DMA_IT_ULE

Update linked-list item error interrupt

DMA_IT_USE

User eetting error interrupt

DMA_IT_SUSP

Completed suspension interrupt

DMA_IT_TO

Trigger overrun interrupt

DMA Priority Level

DMA_LOW_PRIORITY_LOW_WEIGHT

Priority level : Low Priority, Low weight

DMA_LOW_PRIORITY_MID_WEIGHT

Priority level : Low Priority, Mid weight

DMA_LOW_PRIORITY_HIGH_WEIGHT

Priority level : Low Priority, High weight

DMA_HIGH_PRIORITY

Priority level : HIGH Priority

DMA Request Selection

GPDMA1_REQUEST_ADC1

GPDMA1 HW request is ADC1

GPDMA1_REQUEST_ADC4

GPDMA1 HW request is ADC4

GPDMA1_REQUEST_DAC1_CH1

GPDMA1 HW request is DAC1_CH1

GPDMA1_REQUEST_DAC1_CH2
GPDMA1 HW request is DAC1_CH2

GPDMA1_REQUEST_TIM6_UP
GPDMA1 HW request is TIM6_UP

GPDMA1_REQUEST_TIM7_UP
GPDMA1 HW request is TIM7_UP

GPDMA1_REQUEST_SPI1_RX
GPDMA1 HW request is SPI1_RX

GPDMA1_REQUEST_SPI1_TX
GPDMA1 HW request is SPI1_TX

GPDMA1_REQUEST_SPI2_RX
GPDMA1 HW request is SPI2_RX

GPDMA1_REQUEST_SPI2_TX
GPDMA1 HW request is SPI2_TX

GPDMA1_REQUEST_SPI3_RX
GPDMA1 HW request is SPI3_RX

GPDMA1_REQUEST_SPI3_TX
GPDMA1 HW request is SPI3_TX

GPDMA1_REQUEST_I2C1_RX
GPDMA1 HW request is I2C1_RX

GPDMA1_REQUEST_I2C1_TX
GPDMA1 HW request is I2C1_TX

GPDMA1_REQUEST_I2C1_EVC
GPDMA1 HW request is I2C1_EVC

GPDMA1_REQUEST_I2C2_RX
GPDMA1 HW request is I2C2_RX

GPDMA1_REQUEST_I2C2_TX
GPDMA1 HW request is I2C2_TX

GPDMA1_REQUEST_I2C2_EVC
GPDMA1 HW request is I2C2_EVC

GPDMA1_REQUEST_I2C3_RX
GPDMA1 HW request is I2C3_RX

GPDMA1_REQUEST_I2C3_TX
GPDMA1 HW request is I2C3_TX

GPDMA1_REQUEST_I2C3_EVC
GPDMA1 HW request is I2C3_EVC

GPDMA1_REQUEST_I2C4_RX
GPDMA1 HW request is I2C4_RX

GPDMA1_REQUEST_I2C4_TX
GPDMA1 HW request is I2C4_TX

GPDMA1_REQUEST_I2C4_EVC
GPDMA1 HW request is I2C4_EVC

GPDMA1_REQUEST_USART1_RX
GPDMA1 HW request is USART1_RX

GPDMA1_REQUEST_USART1_TX
GPDMA1 HW request is USART1_TX

GPDMA1_REQUEST_USART2_RX
GPDMA1 HW request is USART2_RX

GPDMA1_REQUEST_USART2_TX
GPDMA1 HW request is USART2_TX

GPDMA1_REQUEST_USART3_RX
GPDMA1 HW request is USART3_RX

GPDMA1_REQUEST_USART3_TX
GPDMA1 HW request is USART3_TX

GPDMA1_REQUEST_UART4_RX
GPDMA1 HW request is UART4_RX

GPDMA1_REQUEST_UART4_TX
GPDMA1 HW request is UART4_TX

GPDMA1_REQUEST_UART5_RX
GPDMA1 HW request is UART5_RX

GPDMA1_REQUEST_UART5_TX
GPDMA1 HW request is UART5_TX

GPDMA1_REQUEST_LPUART1_RX
GPDMA1 HW request is LPUART1_RX

GPDMA1_REQUEST_LPUART1_TX
GPDMA1 HW request is LPUART1_TX

GPDMA1_REQUEST_SAI1_A
GPDMA1 HW request is SAI1_A

GPDMA1_REQUEST_SAI1_B
GPDMA1 HW request is SAI1_B

GPDMA1_REQUEST_SAI2_A
GPDMA1 HW request is SAI2_A

GPDMA1_REQUEST_SAI2_B
GPDMA1 HW request is SAI2_B

GPDMA1_REQUEST_OCTOSPI1
GPDMA1 HW request is OCTOSPI1

GPDMA1_REQUEST_OCTOSPI2
GPDMA1 HW request is OCTOSPI2

GPDMA1_REQUEST_TIM1_CH1
GPDMA1 HW request is TIM1_CH1

GPDMA1_REQUEST_TIM1_CH2
GPDMA1 HW request is TIM1_CH2

GPDMA1_REQUEST_TIM1_CH3
GPDMA1 HW request is TIM1_CH3

GPDMA1_REQUEST_TIM1_CH4
GPDMA1 HW request is TIM1_CH4

GPDMA1_REQUEST_TIM1_UP
GPDMA1 HW request is TIM1_UP

GPDMA1_REQUEST_TIM1_TRIG
GPDMA1 HW request is TIM1_TRIG

GPDMA1_REQUEST_TIM1_COM
GPDMA1 HW request is TIM1_COM

GPDMA1_REQUEST_TIM8_CH1
GPDMA1 HW request is TIM8_CH1

GPDMA1_REQUEST_TIM8_CH2
GPDMA1 HW request is TIM8_CH2

GPDMA1_REQUEST_TIM8_CH3
GPDMA1 HW request is TIM8_CH3

GPDMA1_REQUEST_TIM8_CH4
GPDMA1 HW request is TIM8_CH4

GPDMA1_REQUEST_TIM8_UP
GPDMA1 HW request is TIM8_UP

GPDMA1_REQUEST_TIM8_TRIG
GPDMA1 HW request is TIM8_TRIG

GPDMA1_REQUEST_TIM8_COM
GPDMA1 HW request is TIM8_COM

GPDMA1_REQUEST_TIM2_CH1
GPDMA1 HW request is TIM2_CH1

GPDMA1_REQUEST_TIM2_CH2
GPDMA1 HW request is TIM2_CH2

GPDMA1_REQUEST_TIM2_CH3
GPDMA1 HW request is TIM2_CH3

GPDMA1_REQUEST_TIM2_CH4
GPDMA1 HW request is TIM2_CH4

GPDMA1_REQUEST_TIM2_UP
GPDMA1 HW request is TIM2_UP

GPDMA1_REQUEST_TIM3_CH1
GPDMA1 HW request is TIM3_CH1

GPDMA1_REQUEST_TIM3_CH2
GPDMA1 HW request is TIM3_CH2

GPDMA1_REQUEST_TIM3_CH3
GPDMA1 HW request is TIM3_CH3

GPDMA1_REQUEST_TIM3_CH4
GPDMA1 HW request is TIM3_CH4

GPDMA1_REQUEST_TIM3_UP
GPDMA1 HW request is TIM3_UP

GPDMA1_REQUEST_TIM3_TRIG
GPDMA1 HW request is TIM3_TRIG

GPDMA1_REQUEST_TIM4_CH1
GPDMA1 HW request is TIM4_CH1

GPDMA1_REQUEST_TIM4_CH2
GPDMA1 HW request is TIM4_CH2

GPDMA1_REQUEST_TIM4_CH3
GPDMA1 HW request is TIM4_CH3

GPDMA1_REQUEST_TIM4_CH4
GPDMA1 HW request is TIM4_CH4

GPDMA1_REQUEST_TIM4_UP
GPDMA1 HW request is TIM4_UP

GPDMA1_REQUEST_TIM5_CH1
GPDMA1 HW request is TIM5_CH1

GPDMA1_REQUEST_TIM5_CH2
GPDMA1 HW request is TIM5_CH2

GPDMA1_REQUEST_TIM5_CH3
GPDMA1 HW request is TIM5_CH3

GPDMA1_REQUEST_TIM5_CH4
GPDMA1 HW request is TIM5_CH4

GPDMA1_REQUEST_TIM5_UP
GPDMA1 HW request is TIM5_UP

GPDMA1_REQUEST_TIM5_TRIG
GPDMA1 HW request is TIM5_TRIG

GPDMA1_REQUEST_TIM15_CH1
GPDMA1 HW request is TIM15_CH1

GPDMA1_REQUEST_TIM15_UP
GPDMA1 HW request is TIM15_UP

GPDMA1_REQUEST_TIM15_TRIG
GPDMA1 HW request is TIM15_TRIG

GPDMA1_REQUEST_TIM15_COM
GPDMA1 HW request is TIM15_COM

GPDMA1_REQUEST_TIM16_CH1
GPDMA1 HW request is TIM16_CH1

GPDMA1_REQUEST_TIM16_UP
GPDMA1 HW request is TIM16_UP

GPDMA1_REQUEST_TIM17_CH1
GPDMA1 HW request is TIM17_CH1

GPDMA1_REQUEST_TIM17_UP
GPDMA1 HW request is TIM17_UP

GPDMA1_REQUEST_DCMI_PSSI
GPDMA1 HW request is DCMI_PSSI

GPDMA1_REQUEST_AES_IN
GPDMA1 HW request is AES_IN

GPDMA1_REQUEST_AES_OUT
GPDMA1 HW request is AES_OUT

GPDMA1_REQUEST_HASH_IN
GPDMA1 HW request is HASH_IN

GPDMA1_REQUEST_UCPD1_TX
GPDMA1 HW request is UCPD1_TX

GPDMA1_REQUEST_UCPD1_RX
GPDMA1 HW request is UCPD1_RX

GPDMA1_REQUEST_MDF1_FLT0
GPDMA1 HW request is MDF1_FLT0

GPDMA1_REQUEST_MDF1_FLT1
GPDMA1 HW request is MDF1_FLT1

GPDMA1_REQUEST_MDF1_FLT2
GPDMA1 HW request is MDF1_FLT2

GPDMA1_REQUEST_MDF1_FLT3
GPDMA1 HW request is MDF1_FLT3

GPDMA1_REQUEST_MDF1_FLT4
GPDMA1 HW request is MDF1_FLT4

GPDMA1_REQUEST_MDF1_FLT5
GPDMA1 HW request is MDF1_FLT5

GPDMA1_REQUEST_ADF1_FLT0
GPDMA1 HW request is ADF1_FLT0

GPDMA1_REQUEST_FMAC_READ
GPDMA1 HW request is FMAC_READ

GPDMA1_REQUEST_FMAC_WRITE
GPDMA1 HW request is FMAC_WRITE

GPDMA1_REQUEST_CORDIC_READ
GPDMA1 HW request is CORDIC_READ

GPDMA1_REQUEST_CORDIC_WRITE
GPDMA1 HW request is CORDIC_WRITE

GPDMA1_REQUEST_SAES_IN
GPDMA1 HW request is SAES_IN

GPDMA1_REQUEST_SAES_OUT
GPDMA1 HW request is SAES_OUT

GPDMA1_REQUEST_LPTIM1_IC1
GPDMA1 HW request is LPTIM1_IC1

GPDMA1_REQUEST_LPTIM1_IC2
GPDMA1 HW request is LPTIM1_IC2

GPDMA1_REQUEST_LPTIM1_UE
GPDMA1 HW request is LPTIM1_UE

GPDMA1_REQUEST_LPTIM2_IC1
GPDMA1 HW request is LPTIM2_IC1

GPDMA1_REQUEST_LPTIM2_IC2
GPDMA1 HW request is LPTIM2_IC2

GPDMA1_REQUEST_LPTIM2_UE
GPDMA1 HW request is LPTIM2_UE

GPDMA1_REQUEST_LPTIM3_IC1
GPDMA1 HW request is LPTIM3_IC1

GPDMA1_REQUEST_LPTIM3_IC2
GPDMA1 HW request is LPTIM3_IC2

GPDMA1_REQUEST_LPTIM3_UE
GPDMA1 HW request is LPTIM3_UE

GPDMA1_REQUEST_HSPI1
GPDMA1 HW request is HSPI1

GPDMA1_REQUEST_I2C5_RX
GPDMA1 HW request is I2C5_RX

GPDMA1_REQUEST_I2C5_TX
GPDMA1 HW request is I2C5_TX

GPDMA1_REQUEST_I2C5_EVC
GPDMA1 HW request is I2C5_EVC

GPDMA1_REQUEST_I2C6_RX
GPDMA1 HW request is I2C6_RX

GPDMA1_REQUEST_I2C6_TX
GPDMA1 HW request is I2C6_TX

GPDMA1_REQUEST_I2C6_EVC
GPDMA1 HW request is I2C6_EVC

GPDMA1_REQUEST_USART6_RX
GPDMA1 HW request is USART6_RX

GPDMA1_REQUEST_USART6_TX
GPDMA1 HW request is USART6_TX

GPDMA1_REQUEST_ADC2
GPDMA1 HW request is ADC2

GPDMA1_REQUEST_JPEG_RX
GPDMA1 HW request is JPEG_TX

GPDMA1_REQUEST_JPEG_TX
GPDMA1 HW request is JPEG_RX

LPDMA1_REQUEST_LPUART1_RX
LPDMA1 HW request is LPUART1_RX

LPDMA1_REQUEST_LPUART1_TX
LPDMA1 HW request is LPUART1_TX

LPDMA1_REQUEST_SPI3_RX
LPDMA1 HW request is SPI3_RX

LPDMA1_REQUEST_SPI3_TX
LPDMA1 HW request is SPI3_TX

LPDMA1_REQUEST_I2C3_RX
LPDMA1 HW request is I2C3_RX

LPDMA1_REQUEST_I2C3_TX
LPDMA1 HW request is I2C3_TX

LPDMA1_REQUEST_I2C3_EVC
LPDMA1 HW request is I2C3_EVC

LPDMA1_REQUEST_ADC4
LPDMA1 HW request is ADC4

LPDMA1_REQUEST_DAC1_CH1
LPDMA1 HW request is DAC1_CH1

LPDMA1_REQUEST_DAC1_CH2
LPDMA1 HW request is DAC1_CH2

LPDMA1_REQUEST_ADF1_FLT0
LPDMA1 HW request is ADF1_FLT0

LPDMA1_REQUEST_LPTIM1_IC1
LPDMA1 HW request is LPTIM1_IC1

LPDMA1_REQUEST_LPTIM1_IC2
LPDMA1 HW request is LPTIM1_IC2

LPDMA1_REQUEST_LPTIM1_UE
LPDMA1 HW request is LPTIM1_UE

LPDMA1_REQUEST_LPTIM3_IC1
LPDMA1 HW request is LPTIM3_IC1

LPDMA1_REQUEST_LPTIM3_IC2
LPDMA1 HW request is LPTIM3_IC2

LPDMA1_REQUEST_LPTIM3_UE
LPDMA1 HW request is LPTIM3_UE

DMA_REQUEST_SW
DMA SW request

DMA Source Data Width

DMA_SRC_DATAWIDTH_BYTE
Source data width : Byte

DMA_SRC_DATAWIDTH_HALFWORD
Source data width : HalfWord

DMA_SRC_DATAWIDTH_WORD
Source data width : Word

DMA Source Increment Mode

DMA_SINC_FIXED
Source fixed single / burst

DMA_SINC_INCREMENTED
Source incremented single / burst

DMA Transfer Allocated Port

DMA_SRC_ALLOCATED_PORT0
Source allocated Port 0

DMA_SRC_ALLOCATED_PORT1
Source allocated Port 1

DMA_DEST_ALLOCATED_PORT0
Destination allocated Port 0

DMA_DEST_ALLOCATED_PORT1
Destination allocated Port 1

DMA Transfer Direction

DMA_PERIPH_TO_MEMORY

Peripheral to memory direction

DMA_MEMORY_TO_PERIPH

Memory to peripheral direction

DMA_MEMORY_TO_MEMORY

Memory to memory direction

DMA Transfer Event Mode

DMA_TCEM_BLOCK_TRANSFER

The TC event is generated at the end of each block and the HT event is generated at the half of each block

DMA_TCEM_REPEATED_BLOCK_TRANSFER

The TC event is generated at the end of the repeated block and the HT event is generated at the half of the repeated block

DMA_TCEM_EACH_LL_ITEM_TRANSFER

The TC event is generated at the end of each linked-list item and the HT event is generated at the half of each linked-list item

DMA_TCEM_LAST_LL_ITEM_TRANSFER

The TC event is generated at the end of the last linked-list item and the HT event is generated at the half of the last linked-list item

DMA Transfer Mode

DMA_NORMAL

Normal DMA transfer

22 HAL DMA Extension Driver

22.1 DMAEx Firmware driver registers structures

22.1.1 DMA_DataHandlingConfTypeDef

DMA_DataHandlingConfTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***uint32_t DataExchange***
- ***uint32_t DataAlignment***

Field Documentation

- ***uint32_t DMA_DataHandlingConfTypeDef::DataExchange***
  Specifies the DMA channel data exchange mode. This parameter can be a value of ***DMAEx_Data_Exchange***
- ***uint32_t DMA_DataHandlingConfTypeDef::DataAlignment***
  Specifies the DMA channel data padding and alignment mode This parameter can be a value of ***DMAEx_Data_Alignment***

22.1.2 DMA_TriggerConfTypeDef

DMA_TriggerConfTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***uint32_t TriggerMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***

Field Documentation

- ***uint32_t DMA_TriggerConfTypeDef::TriggerMode***
  Specifies the DMA channel trigger mode. This parameter can be a value of ***DMAEx_Trigger_Mode***
- ***uint32_t DMA_TriggerConfTypeDef::TriggerPolarity***
  Specifies the DMA channel trigger event polarity. This parameter can be a value of ***DMAEx_Trigger_Polarity***
- ***uint32_t DMA_TriggerConfTypeDef::TriggerSelection***
  Specifies the DMA channel trigger event selection. This parameter can be a value of ***DMAEx_Trigger_Selection***

22.1.3 DMA_RepeatBlockConfTypeDef

DMA_RepeatBlockConfTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***uint32_t RepeatCount***
- ***int32_t SrcAddrOffset***
- ***int32_t DestAddrOffset***
- ***int32_t BlkSrcAddrOffset***
- ***int32_t BlkDestAddrOffset***

Field Documentation

- ***uint32_t DMA_RepeatBlockConfTypeDef::RepeatCount***
  Specifies the DMA channel repeat count (the number of repetitions of block). This parameter can be a value between 1 and 2048
- ***int32_t DMA_RepeatBlockConfTypeDef::SrcAddrOffset***
  Specifies the DMA channel single/burst source address offset : This parameter can be a value between -8191 and 8191. If source address offset > 0 => Increment the source address by offset from where the last single/burst transfer ends. If source address offset < 0 => Decrement the source address by offset from where the last single/burst transfer ends. If source address offset == 0 => The next single/burst source address starts from where the last transfer ends

- ***int32_t DMA_RepeatBlockConfTypeDef::DestAddrOffset***
  Specifies the DMA channel single/burst destination address offset signed value : This parameter can be a value between -8191 and 8191. If destination address offset > 0 => Increment the destination address by offset from where the last single/burst transfer ends. If destination address offset < 0 => Decrement the destination address by offset from where the last single/burst transfer ends. If destination address offset == 0 => The next single/burst destination address starts from where the last transfer ends.
- ***int32_t DMA_RepeatBlockConfTypeDef::BlkSrcAddrOffset***
  Specifies the DMA channel block source address offset signed value : This parameter can be a value between -65535 and 65535. If block source address offset > 0 => Increment the block source address by offset from where the last block ends. If block source address offset < 0 => Decrement the next block source address by offset from where the last block ends. If block source address offset == 0 => the next block source address starts from where the last block ends
- ***int32_t DMA_RepeatBlockConfTypeDef::BlkDestAddrOffset***
  Specifies the DMA channel block destination address offset signed value : This parameter can be a value between -65535 and 65535. If block destination address offset > 0 => Increment the block destination address by offset from where the last block ends. If block destination address offset < 0 => Decrement the next block destination address by offset from where the last block ends. If block destination address offset == 0 => the next block destination address starts from where the last block ends

22.1.4 DMA_NodeConfTypeDef

DMA_NodeConfTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***uint32_t NodeType***
- ***DMA_InitTypeDef Init***
- ***DMA_DataHandlingConfTypeDef DataHandlingConfig***
- ***DMA_TriggerConfTypeDef TriggerConfig***
- ***DMA_RepeatBlockConfTypeDef RepeatBlockConfig***
- ***uint32_t SrcAddress***
- ***uint32_t DstAddress***
- ***uint32_t DataSize***
- ***uint32_t SrcSecure***
- ***uint32_t DestSecure***

Field Documentation

- ***uint32_t DMA_NodeConfTypeDef::NodeType***
  Specifies the DMA channel node type. This parameter can be a value of ***DMAEx_Node_Type***
- ***DMA_InitTypeDef DMA_NodeConfTypeDef::Init***
  Specifies the DMA channel basic configuration
- ***DMA_DataHandlingConfTypeDef DMA_NodeConfTypeDef::DataHandlingConfig***
  Specifies the DMA channel data handling channel configuration
- ***DMA_TriggerConfTypeDef DMA_NodeConfTypeDef::TriggerConfig***
  Specifies the DMA channel trigger configuration
- ***DMA_RepeatBlockConfTypeDef DMA_NodeConfTypeDef::RepeatBlockConfig***
  Specifies the DMA channel repeated block configuration
- ***uint32_t DMA_NodeConfTypeDef::SrcAddress***
  Specifies the source memory address
- ***uint32_t DMA_NodeConfTypeDef::DstAddress***
  Specifies the destination memory address
- ***uint32_t DMA_NodeConfTypeDef::DataSize***
  Specifies the source data size in bytes
- ***uint32_t DMA_NodeConfTypeDef::SrcSecure***
  Specifies the source security attribute
- ***uint32_t DMA_NodeConfTypeDef::DestSecure***
  Specifies the destination security attribute

22.1.5 DMA_NodeTypeDef

DMA_NodeTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***uint32_t LinkRegisters***
- ***uint32_t NodeInfo***

Field Documentation

- ***uint32_t DMA_NodeTypeDef::LinkRegisters[8U]***
  Physical Node register description
- ***uint32_t DMA_NodeTypeDef::NodeInfo***
  Node information

22.1.6 __DMA_QListTypeDef

__DMA_QListTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***DMA_NodeTypeDef * Head***
- ***DMA_NodeTypeDef * FirstCircularNode***
- ***uint32_t NodeNumber***
- ***__IO HAL_DMA_QStateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Type***

Field Documentation

- ***DMA_NodeTypeDef* __DMA_QListTypeDef::Head***
  Specifies the queue head node
- ***DMA_NodeTypeDef* __DMA_QListTypeDef::FirstCircularNode***
  Specifies the queue first circular node
- ***uint32_t __DMA_QListTypeDef::NodeNumber***
  Specifies the queue node number
- ***__IO HAL_DMA_QStateTypeDef __DMA_QListTypeDef::State***
  Specifies the queue state
- ***__IO uint32_t __DMA_QListTypeDef::ErrorCode***
  Specifies the queue error code
- ***__IO uint32_t __DMA_QListTypeDef::Type***
  Specifies whether the queue is static or dynamic

22.1.7 DMA_NodeInQInfoTypeDef

DMA_NodeInQInfoTypeDef is defined in the stm32u5xx_hal_dma_ex.h

Data Fields

- ***uint32_t cllr_offset***
- ***uint32_t previousnode_addr***
- ***uint32_t currentnode_pos***
- ***uint32_t currentnode_addr***
- ***uint32_t nextnode_addr***

Field Documentation

- ***uint32_t DMA_NodeInQInfoTypeDef::cllr_offset***
- ***uint32_t DMA_NodeInQInfoTypeDef::previousnode_addr***
- ***uint32_t DMA_NodeInQInfoTypeDef::currentnode_pos***
- ***uint32_t DMA_NodeInQInfoTypeDef::currentnode_addr***
- ***uint32_t DMA_NodeInQInfoTypeDef::nextnode_addr***

22.2 DMAEx Firmware driver API description

The following section lists the various functions of the DMAEx library.

22.2.1 How to use this driver

Alternatively to the normal programming mode, a DMA channel can be programmed by a list of transfers, known as linked-list (list of Node items). Each node is defined by its data structure. Each node specifies a standalone DMA channel. When enabled, the DMA channel fetch the first linked-list node from SRAM (known as head node). When executed, the next linked list node will be fetched and executed. This operation is repeated until the end of the whole linked-list queue. Optionally, the linked-list can be linear where the last linked-list queue node is not linked to another queue node or circular where the last linked-list node is linked to any linked-list queue node.

- Linear linked-list: The DMA channel fetch and execute all DMA linked-list queue from first node (head node) to last node (tail node) ones. When the last node is completed, the DMA channel remains in idle state and another transfer can be lunched.
- Circular linked-list: The DMA channel fetch and execute all DMA linked-list queue from first node (head node) to last node (tail node). When last node is executed, the DMA channel fetches the first circular node another time and repeat the same sequence in an infinite loop (Circular transfer). To stop the DMA channel, an abort operation is required. This linked-list mode replaces the legacy circular transfers.

In order to reduce linked-list queue executing time and power consumption, the DMA channel supports executing the dynamic linked-list format. In fact, the DMA supports the execution of 2 types of linked-list formats : static and dynamic.

- Static linked-list: The static linked-list format refers to the full linked-list node where all DMA channel parameters are fetched and executed independently of the redundancy of information.
- Dynamic linked-list: The dynamic linked-list format refer to the customized linked-list node where only DMA channel necessary parameters are fetched and executed (Example: data size = 20 on previous node, and data size = 20 on the current node => No need to update it). For linked-list transfers, the DMA channel can execute the linked-list queue node by node. This feature is named link step mode. When activated, enabling the DMA channel first time allows to fetch the head node from memory then it stops. Then, another DMA channel enable is needed to execute the node. After that, keeping enabling the DMA channel is needed to execute each node until the end of linked-list queue. When the linked-list queue is circular, enabling the DMA channel in an infinite loop is required to keep the DMA channel running. This feature is useful for debug purpose or asynchronously executing queue nodes.

Each DMA channel transfer (normal or linked-list), is highly configurable according to DMA channel instance integrated in devices. These configuration can be :

- Repeated block configuration : If the feature is supported, the DMA channel can performs a repeated block transfers. Named also 2 dimension addressing transfers, this feature can transfer n iteration of programmed block transfer (Block transfer is the legacy data size). Additional to the repeat count of a block, DMA channel addresses can jump after at burst and block level. The jump length is a programmable parameter defined by DMA user.
  - Jump at burst level : The DMA channel keep an empty area, between each 2 consecutive bursts transmitted.
  - Jump at block level : The DMA channel keep an empty area, between each 2 consecutive blocks transmitted.
- Trigger : The DMA channel transfers can be conditioned by hardware signals edges (rising or falling) named hardware triggers. Trigger condition can be applied at :
  - Single/Burst level : Each single/burst data transmission is conditioned by a signal trigger hit.
  - Block level : Each block data transmission is conditioned by a signal trigger hit.
  - Repeated block level : Each repeated block data transmission is conditioned by a signal trigger hit.
  - Node level : Each node execution is conditioned by a signal trigger hit. The DMA channel can report a trigger overrun when detects more than 2 trigger signal edges before executing the current transfer.
- Data handling : The data handling feature is a FIFO capability that can be :
  - Padding pattern : Padding selected pattern (zero padding or sign extension) when the source data width is smaller than the destination data width at single level.
  - Truncation : Truncate section from the source data single when the source data width is bigger than the destination data width.
  - Pack/Unpack : Pack a set of data when source data width is smaller than the destination data width. Unpack a set of data when source data width is bigger than the destination data width.
  - Exchange : Exchange data at byte and half-word on the destination and at byte level on the source.

Each DMA channel transfer (normal or linked-list) when it is active, can be suspended and resumed at run time application. When trying to suspend an ongoing transfer, the DMA channel isn't suspended instantly but complete the current ongoing single/burst then it stops. When the DMA channel is suspended, the current transfer can be resumed instantly.

The DMA channel that supports FIFO, can report in real time the number of beats remains on destination (Output) FIFO level.

Linked-List Initialization and De-Initialization operation

Differently from normal transfers, DMA channel initialization and de-initialization need less parameters as the remaining transfer parameters are defined by linked-list nodes.

- Use HAL_DMAEx_List_Init() to initialize a DMA channel in linked-list mode according to programmed fields. When called, the DMA channel will be ready to execute linked-list queues.
- Use HAL_DMAEx_List_DeInit() to de-initialize a DMA channel in linked-list mode. When called, the DMA channel will be in reset. It is mandatory to reinitialize it for next transfer.

Linked-List I/O Operation

- Use HAL_DMAEx_List_Start() to start a DMA transfer in linked-list mode after the configuration of linked-list queue base address and offset in polling mode (Blocking mode).
- Use HAL_DMAEx_List_Start_IT() to start a DMA transfer in linked-list mode after the configuration of linked-list queue base address and offset in interrupt mode (Non-blocking mode).

Linked-List Management

The linked-list management is a software processing independently of DMA channel hardware. It allows to reset, build, create, insert, remove, replace, circularize, convert both nodes and queue in order to perform DMA channel transfers in linked-list mode. Linked-list APIs and types are adapted to reduce memory footprint.

Linked-list nodes building

At node level, the operations that can be done are building a new linked-list node or get a linked-list node information from a built node. The linked-list nodes have two forms according to 2 dimensions addressing capability. The linear addressing nodes contains the information of all DMA channel features except the 2 dimension addressing features and the 2 dimensions addressing nodes contain the information of all available features.

- Use HAL_DMAEx_List_BuildNode() to build the DMA linked-list node according to the specified parameters. Build operation allow to convert the specified parameter in values known by the DMA channel and place them in memory. Placing DMA linked-list in SRAM must be done in accordance to product specification to ensure that the link access port can access to the specified SRAM.
  - The DMA linked-list node parameter address should be 32bit aligned and should not exceed the 64 KByte addressable space.
- Use HAL_DMAEx_List_GetNodeConfig() to get the specified configuration parameter on building node. This API can be used when need to change few parameter to build new node.

Inserting nodes to linked-list queue

In order to build a sequence of DMA transaction with different configuration, we need to insert built node at linked-list queue (node present an elementary DMA transaction) in linked-list queue on any position to have the full flexibility of ordering nodes or extend the sequence of queue transactions.

- Use HAL_DMAEx_List_InsertNode() to insert new built node in any queue position of linked-list queue according to selecting previous node. When calling this API with previous node parameter is NULL, the inserted node will be placed at the head of the linked-list queue.
  - This API must be used after HAL_DMAEx_List_BuildNode() otherwise an error will be returned.
  - This API must be called for static queues format.
  - This API shall be avoided when adding new node at the head or the tail of queue (overhead of footprint and performance : use HAL_DMAEx_List_InsertNode_Head() or HAL_DMAEx_List_InsertNode_Tail() instead).

- Use HAL_DMAEx_List_InsertNode_Head() to insert new built node at the head of linked-list queue. The head node will not be overwritten but will be the second queue node.
  - This API must be used after HAL_DMAEx_List_BuildNode() otherwise an error will be returned.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_InsertNode_Tail() to insert new built node at the tail of linked-list queue. The tail node will not be overwritten but will be the penultimate queue node.
  - This API must be used after HAL_DMAEx_List_BuildNode() otherwise an error will be returned.
  - This API must be called for static queues format.

Removing nodes from linked-list queue

There is some cases when removing a node from linked-list queue is needed (need to remove an elementary DMA transaction). Removing node allows to unlink a node from DMA linked-list queue (NOT DELETED), so the removed node can be reused for another queue or to be added to the same queue without need to rebuild it in next step.

- Use HAL_DMAEx_List_RemoveNode() to remove any yet built and inserted node from linked-list queue according to selected node.
  - This API must be called for static queues format.
  - This API shall be avoided when removing the head or the tail of linked-list queue (overhead of footprint and performance : use HAL_DMAEx_List_RemoveNode_Head() or HAL_DMAEx_List_RemoveNode_Tail() instead).
- Use HAL_DMAEx_List_RemoveNode_Head() to remove the head node from linked-list queue.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_RemoveNode_Tail() to remove the tail node from linked-list queue.
  - This API must be called for static queues format.

Replacing nodes on linked-list queue

There is some cases when replacing a node from linked-list queue is needed (need to replace an elementary DMA transfer, by another one that have not the same configuration). Replacing node allows to unlink the node to be replaced from DMA linked-list queue (NOT DELETED) and link instead a new node. So the replaced node can be reused for another queue or to be added to the same queue without need to rebuild it in next step and the new node cannot be reused except when remove it or replaced in next step.

- Use HAL_DMAEx_List_ReplaceNode() to replace any yet built and inserted node on linked-list queue according to selected node.
  - This API must be called for static queues format.
  - This API shall be avoided when replacing the head or the tail linked-list queue (overhead of footprint and performance : use HAL_DMAEx_List_ReplaceNode_Head() or HAL_DMAEx_List_ReplaceNode_Tail() instead).
- Use HAL_DMAEx_List_ReplaceNode_Head() to replace the head node of linked-list queue.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_ReplaceNode_Tail() to replace the tail node from linked-list queue.
  - This API must be called for static queues format.

Reset linked-list queue

After finishing using a linked-list queue, it can be reset and cleared and it's content nodes will be unlinked (NOT DELETED) and reused on another queue.

- Use HAL_DMAEx_List_ResetQ() to reset a linked-list queue and unlink all it's content nodes.
  - This API must be called for ready state queues.
  - This API must be called for static queues format.

Inserting linked-list queue

To ensure the flexibility of building linked-list queue by their targeted functionalities (Example: 3 nodes for action 1 and 5 nodes for action 2), it is possible to build a queue for action 1 that contains action 1 nodes and a queue for action 2 that contains action 2 nodes then concatenating the 2 queues. So, there are some cases where the management of linked-list at queue granularity is needed.

- Use HAL_DMAEx_List_InsertQ() to insert source linked-list queue to a destination linked-list queue according to selecting previous node.
  - This API must be called for static queues format.
  - This API shall be avoided when inserting source linked-list queue at the head or the tail of destination queue (overhead of footprint and performance : use HAL_DMAEx_List_InsertQ_Head() or HAL_DMAEx_List_InsertQ_Tail() instead).
- Use HAL_DMAEx_List_InsertQ_Head() to insert a source linked-list queue at the head of linked-list destination queue.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_InsertQ_Tail() to insert a source linked-list queue at the tail of linked-list destination queue.
  - This API must be called for static queues format.

Circularizing linked-list queue

In order to perform tasks in infinite loop with DMA channel, it is possible to circularize the linked-list queues. Circularizing queue allows to link last linked-list queue node to any previous node of the same queue (This node is named first circular queue). When the first circular node is the head node, all linked-list queue nodes will be executed in infinite loop. When the first circular node is not the head nodes, all precedent nodes are executed once and all remaining nodes are executed in an infinite loop.

- Use HAL_DMAEx_List_SetCircularModeConfig() to circularize the linked-list queue according to first circular node selected.
  - This API must be called for static queues format.
  - This API shall be avoided when first circular node is the head linked-list queue node (overhead of footprint and performance : use HAL_DMAEx_List_SetCircularMode() instead).
- Use HAL_DMAEx_List_SetCircularMode() to circularize the linked-list queue with linking last queue node with first queue node.
  - This API must be called for static queues format.
- Use HAL_DMAEx_List_ClearCircularMode() to clear any linked-list queue circular configuration.
  - This API must be called for static queues format.

Converting linked-list queue

To have the best DMA channel linked-list queue execution, it is recommended to convert yet build linked-list queue to dynamic format (Static is the default format). When linked-list queue becomes dynamic, all queue nodes are optimized and only changed parameters will be updated between nodes. So, the DMA will fetch only changes parameters instead of the whole node.

- Use HAL_DMAEx_List_ConvertQToDynamic() to convert a linked-list queue to dynamic format.
  - This API must be called for ready state queues.
  - This API must be called for static queues format.
  - This API must be called as the last API before starting the DMA channel in linked-list mode.
- Use HAL_DMAEx_List_ConvertQToStatic() to convert a linked-list queue to static format.
  - This API must be called for ready state queues.
  - This API must be called for dynamic queues format.
  - This API must be called as the first API after the full execution of linked-list queue when the execution mode is linear (not circular) if it is dynamic and a linked-list queue management is needed.
  - This API must be called as the first API after the aborting the execution of the current linked-list queue when the execution mode is linear (not circular) if it is dynamic and a linked-list queue management is needed.

When converting a circular queue to dynamic format and when the first circular node is the last queue node, it is recommended to duplicate the last circular node in order to ensure the full optimization when calling HAL_DMAEx_List_ConvertQToDynamic() API. In this case, updated information are only addresses which allow to reduce 4 words of update for linear nodes per node execution and 6 words update for 2 dimensions addressing nodes per node execution.

Linking linked-list queue to DMA channel

In order to have the possibility of the creation of an infinity queues (limited by available memory size), the building of linked-list queue is fully independent from DMA channels. It is possible to build all needed queues if their size is less then available memory at startup time, then linking each time when needed a linked-list queue to an idle DMA channel.

- Use HAL_DMAEx_List_LinkQ() to link a ready linked-list queue to ready DMA channel.
  - This API supports the two format of linked-list (Static and dynamic).
  - This API must be called for ready state queues and DMA channels.
- Use HAL_DMAEx_List_ConvertQToStatic() to unlink a ready linked-list queue to ready DMA channel.
  - This API supports the two format of linked-list (Static and dynamic).
  - This API must be called for ready state queues and DMA channels.

User sequence

To use cleanly the DMA linked-list library, ensure to apply the following call sequences :

- Linear transfer : Linked-list queue building
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail() . . .
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail()
  - HAL_DMAEx_List_ConvertQToDynamic() Linked-list queue execution
  - HAL_DMAEx_List_Init()
  - HAL_DMAEx_List_LinkQ()
  - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
  - HAL_DMAEx_List_UnLinkQ()
  - HAL_DMAEx_List_DeInit()
- Circular transfer : Linked-list queue building
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail() . . .
  - HAL_DMAEx_List_BuildNode()
  - HAL_DMAEx_List_InsertNode_Tail()
  - HAL_DMAEx_List_SetCircularModeConfig() / HAL_DMAEx_List_SetCircularMode()
  - HAL_DMAEx_List_ConvertQToDynamic() Linked-list queue execution
  - HAL_DMAEx_List_Init()
  - HAL_DMAEx_List_LinkQ()
  - HAL_DMAEx_List_Start() / HAL_DMAEx_List_Start_IT()
  - HAL_DMA_Abort() / HAL_DMA_Abort_IT()
  - HAL_DMAEx_List_UnLinkQ()
  - HAL_DMAEx_List_DeInit()

Data Handling

In order to avoid some CPU data processing in several cases, the DMA channel provides some features related to FIFO capabilities titled data handling. (++) Padding pattern Padding selected pattern (zero padding or sign extension) when the source data width is smaller than the destination data width at single level. Zero padding (Source : 0xABAB ------> Destination : 0xABAB0000) Sign bit extension (Source : 0x0ABA ------> Destination : 0x00000ABA) (Source : 0xFABA ------> Destination : 0xFFFFFABA) (++) Truncation : Truncate section from the source data single when the source data width is bigger than the destination data width. Left truncation (Source : 0xABABCDCD ------> Destination : 0xCDCD) Right truncation (Source : 0xABABCDCD ------> Destination : 0xABAB) (++) Pack/Unpack : Pack a set of data when source data width is smaller than the destination data width. Unpack a set of data when source data width is bigger than the destination data width. Pack (Source : 0xAB, 0xCD ------> Destination : 0xABCD) UnPack (Source : 0xABCD ------> Destination : 0xAB, 0xCD) (++) Exchange : Exchange data at byte and half-word on the destination and at byte level on the source. Considering source and destination are both word type. Exchange operation can be as follows. In examples below, one exchange setting is enabled at a time. Source byte exchange only (Source : 0xAB12CD34 ------> Destination : 0xABCD1234) Destination byte exchange only (Source : 0xAB12CD34 ------> Destination : 0x12AB34CD) Destination half-word exchange only (Source : 0xAB12CD34 ------> Destination : 0xCD34AB12)

- Use HAL_DMAEx_ConfigDataHandling() to configure data handling features. Previous elementary explained can be combined according to application needs.
  - This API is complementary of normal transfers.
  - This API must not be called for linked-list transfers as data handling information are configured at node level.

User sequence

To configure cleanly the DMA channel data handling, ensure to apply the following call sequence :

- Linear transfer :
  - HAL_DMA_Init()
  - HAL_DMAEx_ConfigDataHandling()
  - HAL_DMA_Start()

Repeated Block

When available, this feature is used when the data size is higher then 65535 bytes (Maximum block size) or for scattering / gathering data. (++) Gather data Source Destination 0xAA 0xAA 0xBB 0xAA 0xAA ==> 0xAA 0xCC 0xAA (++) Scatter data Source Destination 0xAA 0xAA 0xAA 0xBB 0xAA ==> 0xAA 0xBB 0xAA

- Use HAL_DMAEx_ConfigRepeatBlock() to configure data repeated block feature. Jump addresses and incrementing or decrementing on source and destination can be combined to have the need application behavior.
  - This API is complementary of normal transfers.
  - This API must not be called for linked-list transfers as repeated block information are configured at node level.
  - This API must be called only for DMA channel that supports repeated block feature.

User sequence

To configure cleanly the DMA channel repeated block, ensure to apply the following call sequence :

- Linear transfer :
  - HAL_DMA_Init()
  - HAL_DMAEx_ConfigRepeatBlock()
  - HAL_DMA_Start()

Trigger Configuration

When application needs that DMA transfers are conditioned by internal or external events, the trigger feature can do that. Trigger signals are a set of device signal that are linked to DMA trigger inputs that allows to start the DMA transfers. To setup a trigger transfers, three DMA channel parameters are needed:

- Trigger mode This parameter specifies the trig level.
  - Block level
  - Repeated block level
  - Node level
  - Single / Burst level
- Trigger polarity This parameter specifies the DMA trigger sensitivity (Rising or falling).
- Trigger selection This parameter specifies the DMA trigger hardware signal.
- Use HAL_DMAEx_ConfigTrigger() to configure trigger feature.
  - This API is complementary to normal transfers APIs.
  - This API must not be called for linked-list transfers as trigger information are configured at node level.

User sequence

To configure cleanly the DMA channel trigger, ensure to apply the following call sequence :

- Linear transfer :
  - HAL_DMA_Init()
  - HAL_DMAEx_ConfigTrigger()
  - HAL_DMA_Start()

Suspend and resume operation

There are several cases when needs to suspend a DMA current transfer (Example: liberate bandwidth for more priority DMA channel transfer). Suspending DMA channel (same as abort) is available in polling (blocking mode) and interrupt (non-blocking mode) modes. When suspended, a DMA channel can be instantly resumed.

- Use HAL_DMAEx_Suspend() to suspend an ongoing DMA channel transfer in polling mode (Blocking mode).
- Use HAL_DMAEx_Suspend_IT() to suspend an ongoing DMA channel transfer in interrupt mode (Non-blocking mode).
- Use HAL_DMAEx_Resume() to resume a suspended DMA channel transfer execution.

FIFO status

In several cases, the information of FIFO level is useful to inform at application level how to process remaining data. When not empty, the DMA channel FIFO cannot be flashed only by reset.

- Use HAL_DMAEx_GetFifoLevel() to get the DMA channel FIFO level (available beats in FIFO).

22.2.2 Linked-List Initialization and De-Initialization Functions

This section provides functions allowing to initialize and de-initialize the DMA channel in linked-list mode.

- The HAL_DMAEx_List_Init() function follows the DMA channel linked-list mode configuration procedures as described in reference manual.
- The HAL_DMAEx_List_DeInit() function allows to de-initialize the DMA channel in linked-list mode.

This section contains the following APIs:

- *HAL_DMAEx_List_Init()*
- *HAL_DMAEx_List_DeInit()*

22.2.3 Linked-List IO Operation Functions

This section provides functions allowing to :

- Configure to start DMA transfer in linked-list mode.
- The HAL_DMAEx_List_Start() function allows to start the DMA channel transfer in linked-list mode (Blocking mode).
- The HAL_DMAEx_List_Start_IT() function allows to start the DMA channel transfer in linked-list mode (Non-blocking mode).
  - It is mandatory to register a linked-list queue to be executed by a DMA channel before starting transfer otherwise a HAL_ERROR will be returned.

This section contains the following APIs:
- *HAL_DMAEx_List_Start()*
- *HAL_DMAEx_List_Start_IT()*

22.2.4 Linked-List Management Functions

This section provides functions allowing to :
- Build linked-list node.
- Get linked-list node configuration.
- Insert node to linked-list queue in any queue position.
- Remove any node from linked-list queue.
- Replace any node from linked-list queue.
- Reset linked-list queue.
- Insert linked-list queue in any queue position.
- Set circular mode configuration to linked-list queue.
- Clear circular mode configuration from linked-list queue.
- Convert static linked-list queue to dynamic format.
- Convert dynamic linked-list queue to static format.
- Link linked-list queue to DMA channel.
- Unlink linked-list queue from DMA channel.
- The HAL_DMAEx_List_BuildNode() function allows to build linked-list node. Node type can be :
  - 2 dimensions addressing node.
  - Linear addressing node.
- The HAL_DMAEx_List_GetNodeConfig() function allows to get the linked-list node configuration from built node.
- The HAL_DMAEx_List_InsertNode() function allows to insert built linked-list node to static linked-list queue according to selected position.
- The HAL_DMAEx_List_InsertNode_Head() and HAL_DMAEx_List_InsertNode_Tail() functions allow to insert built linked-list node to the head (respectively the tail) of static linked-list queue.
- The HAL_DMAEx_List_RemoveNode() function allows to remove selected built linked-list node from static linked-list queue.
- The HAL_DMAEx_List_RemoveNode_Head() and HAL_DMAEx_List_RemoveNode_Tail() functions allow to remove the head (respectively the tail) built linked-list node from static linked-list queue.
- The HAL_DMAEx_List_ReplaceNode() function allows to replace selected built linked-list node from static linked-list queue.
- The HAL_DMAEx_List_ReplaceNode_Head() and HAL_DMAEx_List_ReplaceNode_Tail() functions allow to replace the head (respectively the tail) built linked-list node of static linked-list queue.
- The HAL_DMAEx_List_ResetQ() function allows to reset static linked-list queue and unlink all built linked-list nodes.
- The HAL_DMAEx_List_InsertQ() function allows to insert static linked-list source queue to static linked-list destination queue according to selected position.
- The HAL_DMAEx_List_InsertQ_Head() and HAL_DMAEx_List_InsertQ_Tail() functions allow to insert static linked-list source queue to the head (respectively the tail) of static linked-list destination queue.
- The HAL_DMAEx_List_SetCircularModeConfig() function allows to link the last static linked-list queue node to the selected first circular node.
- The HAL_DMAEx_List_SetCircularMode() function allows to link the last static linked-list queue node to the first static linked-list queue node.
- The HAL_DMAEx_List_ClearCircularMode() function allows to unlink the last static linked-list queue node from any first circular node position.
- The HAL_DMAEx_List_ConvertQToDynamic() function allows to convert the static linked-list queue to dynamic format. (Optimized queue execution)
- The HAL_DMAEx_List_ConvertQToStatic() function allows to convert the dynamic linked-list queue to static format. (Not optimized queue execution)

- The HAL_DMAEx_List_LinkQ() function allows to link the (Dynamic / Static) linked-list queue to DMA channel to be executed.
- The HAL_DMAEx_List_UnLinkQ() function allows to unlink the (Dynamic / Static) linked-list queue from DMA channel when execution is completed.

This section contains the following APIs:
- *HAL_DMAEx_List_BuildNode()*
- *HAL_DMAEx_List_GetNodeConfig()*
- *HAL_DMAEx_List_InsertNode()*
- *HAL_DMAEx_List_InsertNode_Head()*
- *HAL_DMAEx_List_InsertNode_Tail()*
- *HAL_DMAEx_List_RemoveNode()*
- *HAL_DMAEx_List_RemoveNode_Head()*
- *HAL_DMAEx_List_RemoveNode_Tail()*
- *HAL_DMAEx_List_ReplaceNode()*
- *HAL_DMAEx_List_ReplaceNode_Head()*
- *HAL_DMAEx_List_ReplaceNode_Tail()*
- *HAL_DMAEx_List_ResetQ()*
- *HAL_DMAEx_List_InsertQ()*
- *HAL_DMAEx_List_InsertQ_Head()*
- *HAL_DMAEx_List_InsertQ_Tail()*
- *HAL_DMAEx_List_SetCircularModeConfig()*
- *HAL_DMAEx_List_SetCircularMode()*
- *HAL_DMAEx_List_ClearCircularMode()*
- *HAL_DMAEx_List_ConvertQToDynamic()*
- *HAL_DMAEx_List_ConvertQToStatic()*
- *HAL_DMAEx_List_LinkQ()*
- *HAL_DMAEx_List_UnLinkQ()*

22.2.5 Data handling, repeated block and trigger configuration functions

This section provides functions allowing to :
- Configure DMA channel data handling.
- Configure DMA channel repeated block.
- Configure DMA channel trigger.
- The HAL_DMAEx_ConfigDataHandling() function allows to configure DMA channel data handling.
  - GPDMA data handling : byte-based reordering, packing/unpacking, padding/truncation, sign extension and left/right alignment.
  - LPDMA data handling : byte-based padding/truncation, sign extension and left/right alignment.
- The HAL_DMAEx_ConfigTrigger() function allows to configure DMA channel HW triggers.
- The HAL_DMAEx_ConfigRepeatBlock() function allows to configure DMA channel repeated block.
  - This feature is available only for channel that supports 2 dimensions addressing capability.

This section contains the following APIs:
- *HAL_DMAEx_ConfigDataHandling()*
- *HAL_DMAEx_ConfigTrigger()*
- *HAL_DMAEx_ConfigRepeatBlock()*

22.2.6 Suspend and resume operation functions

This section provides functions allowing to :
- Suspend any ongoing DMA channel transfer.
- Resume any suspended DMA channel transfer.

- The HAL_DMAEx_Suspend() function allows to suspend any ongoing DMA channel transfer in polling mode (Blocking mode).
- The HAL_DMAEx_Suspend_IT() function allows to suspend any ongoing DMA channel transfer in interrupt mode (Non-blocking mode).
- The HAL_DMAEx_Resume() function allows to resume any suspended DMA channel transfer.

This section contains the following APIs:
- *HAL_DMAEx_Suspend()*
- *HAL_DMAEx_Suspend_IT()*
- *HAL_DMAEx_Resume()*

22.2.7 Fifo status function

This section provides function allowing to get DMA channel FIFO level.
- The HAL_DMAEx_GetFifoLevel() function allows to return the number of available write beats in the FIFO, in units of the programmed destination data.
  – This API is available only for DMA channels that supports FIFO.

This section contains the following APIs:
- *HAL_DMAEx_GetFifoLevel()*

22.2.8 Detailed description of functions

HAL_DMAEx_List_Init

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Init (DMA_HandleTypeDef *const hdma)

Function description

Initialize the DMA channel in linked-list mode according to the specified parameters in the DMA_InitLinkedListTypeDef and create the associated handle.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_DeInit

Function name

HAL_StatusTypeDef HAL_DMAEx_List_DeInit (DMA_HandleTypeDef *const hdma)

Function description

DeInitialize the DMA channel when it is configured in linked-list mode.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_Start

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Start (DMA_HandleTypeDef *const hdma)

Function description

Start the DMA channel transfer in linked-list mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_Start_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_List_Start_IT (DMA_HandleTypeDef *const hdma)

Function description

Starts the DMA channel transfer in linked-list mode with interrupts enabled (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_List_BuildNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_BuildNode (DMA_NodeConfTypeDef const *const pNodeConfig, DMA_NodeTypeDef *const pNode)

Function description

Build a DMA channel node according to the specified parameters in the DMA_NodeConfTypeDef.

Parameters

- **pNodeConfig:** : Pointer to a DMA_NodeConfTypeDef structure that contains the configuration information for the specified DMA linked-list Node.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list node registers configurations.

Return values

- **HAL:** status.

Notes

- The DMA linked-list node parameter address should be 32bit aligned and should not exceed the 64 KByte addressable space.

HAL_DMAEx_List_GetNodeConfig

Function name

HAL_StatusTypeDef HAL_DMAEx_List_GetNodeConfig (DMA_NodeConfTypeDef *const pNodeConfig, DMA_NodeTypeDef const *const pNode)

Function description

Get a DMA channel node configuration.

Parameters

- **pNodeConfig:** : Pointer to a DMA_NodeConfTypeDef structure that contains the configuration information for the specified DMA linked-list Node.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pPrevNode, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node in any queue position of linked-list queue according to selecting previous node.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pPrevNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode_Head (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node at the head of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertNode_Tail (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Insert new node at the tail of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNode)

Function description

Remove node from any linked-list queue position.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode_Head (DMA_QListTypeDef *const pQList)

Function description

Remove the head node from linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_RemoveNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_RemoveNode_Tail (DMA_QListTypeDef *const pQList)

Function description

Remove the tail node from linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pOldNode, DMA_NodeTypeDef *const pNewNode)

Function description

Replace node in linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pOldNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list old node registers configurations.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode_Head (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Replace the head node of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ReplaceNode_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ReplaceNode_Tail (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pNewNode)

Function description

Replace the tail node of linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pNewNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list new node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_ResetQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ResetQ (DMA_QListTypeDef *const pQList)

Function description

Reset the linked-list queue and unlink queue nodes.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ (DMA_QListTypeDef *const pSrcQList, DMA_NodeTypeDef const *const pPrevNode, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue to a destination linked-list queue according to selecting previous node.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pPrevNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list previous node registers configurations.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ_Head

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ_Head (DMA_QListTypeDef *const pSrcQList, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue at the head of destination queue.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_InsertQ_Tail

Function name

HAL_StatusTypeDef HAL_DMAEx_List_InsertQ_Tail (DMA_QListTypeDef *const pSrcQList, DMA_QListTypeDef *const pDestQList)

Function description

Insert a source linked-list queue at the tail of destination queue.

Parameters

- **pSrcQList:** : Pointer to a DMA_QListTypeDef structure that contains source queue information.
- **pDestQList:** : Pointer to a DMA_QListTypeDef structure that contains destination queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_SetCircularModeConfig

Function name

HAL_StatusTypeDef HAL_DMAEx_List_SetCircularModeConfig (DMA_QListTypeDef *const pQList, DMA_NodeTypeDef *const pFirstCircularNode)

Function description

Set circular mode configuration for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.
- **pFirstCircularNode:** : Pointer to a DMA_NodeTypeDef structure that contains linked-list first circular node registers configurations.

Return values

- **HAL:** status.

HAL_DMAEx_List_SetCircularMode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_SetCircularMode (DMA_QListTypeDef *const pQList)

Function description

Set circular mode for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ClearCircularMode

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ClearCircularMode (DMA_QListTypeDef *const pQList)

Function description

Clear circular mode for linked-list queue.

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ConvertQToDynamic

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ConvertQToDynamic (DMA_QListTypeDef *const pQList)

Function description

Convert a linked-list queue to dynamic (Optimized DMA queue execution).

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_ConvertQToStatic

Function name

HAL_StatusTypeDef HAL_DMAEx_List_ConvertQToStatic (DMA_QListTypeDef *const pQList)

Function description

Convert a linked-list queue to static (Not optimized DMA queue execution).

Parameters

- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_LinkQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_LinkQ (DMA_HandleTypeDef *const hdma, DMA_QListTypeDef *const pQList)

Function description

Link linked-list queue to a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pQList:** : Pointer to a DMA_QListTypeDef structure that contains queue information.

Return values

- **HAL:** status.

HAL_DMAEx_List_UnLinkQ

Function name

HAL_StatusTypeDef HAL_DMAEx_List_UnLinkQ (DMA_HandleTypeDef *const hdma)

Function description

Unlink linked-list queue from a DMA channel.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigDataHandling

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigDataHandling (DMA_HandleTypeDef *const hdma, DMA_DataHandlingConfTypeDef const *const pConfigDataHandling)

Function description

Configure the DMA channel data handling according to the specified parameters in the DMA_DataHandlingConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigDataHandling:** : Pointer to a DMA_DataHandlingConfTypeDef structure that contains the data handling configuration.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigTrigger

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigTrigger (DMA_HandleTypeDef *const hdma, DMA_TriggerConfTypeDef const *const pConfigTrigger)

Function description

Configure the DMA channel trigger according to the specified parameters in the DMA_TriggerConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigTrigger:** : Pointer to a DMA_TriggerConfTypeDef structure that contains the trigger configuration.

Return values

- **HAL:** status.

HAL_DMAEx_ConfigRepeatBlock

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigRepeatBlock (DMA_HandleTypeDef *const hdma, DMA_RepeatBlockConfTypeDef const *const pConfigRepeatBlock)

Function description

Configure the DMA channel repeated block according to the specified parameters in the DMA_RepeatBlockConfTypeDef.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **pConfigRepeatBlock:** : Pointer to a DMA_RepeatBlockConfTypeDef structure that contains the repeated block configuration.

Return values

- **HAL:** status.

HAL_DMAEx_Suspend

Function name

HAL_StatusTypeDef HAL_DMAEx_Suspend (DMA_HandleTypeDef *const hdma)

Function description

Suspend any ongoing DMA channel transfer in polling mode (Blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status.

Notes

- After suspending a DMA channel, a check for wait until the DMA channel is effectively suspended is added. If a channel is suspended while a data transfer is ongoing, the current data will be transferred and the channel will be effectively suspended only after the transfer of this single/burst data is finished.

HAL_DMAEx_Suspend_IT

Function name

HAL_StatusTypeDef HAL_DMAEx_Suspend_IT (DMA_HandleTypeDef *const hdma)

Function description

Suspend any ongoing DMA channel transfer in polling mode (Non-blocking mode).

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_Resume

Function name

HAL_StatusTypeDef HAL_DMAEx_Resume (DMA_HandleTypeDef *const hdma)

Function description

Resume any suspended DMA channel transfer.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status.

HAL_DMAEx_GetFifoLevel

Function name

uint32_t HAL_DMAEx_GetFifoLevel (DMA_HandleTypeDef const *const hdma)

Function description

Get and returns the DMA channel FIFO level.

Parameters

- **hdma:** : Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **Returns:** the number of available beats in FIFO.

22.3 DMAEx Firmware driver defines

The following section lists the various define and macros of the module.

22.3.1 DMAEx

DMAEx

DMAEx Data Alignment

DMA_DATA_RIGHTALIGN_ZEROPADDED

If source data width < destination data width => Right aligned padded with 0 up to destination data width

DMA_DATA_RIGHTALIGN_LEFTTRUNC

If source data width > destination data width => Right aligned left Truncated down to destination data width

DMA_DATA_RIGHTALIGN_SIGNEXT

If source data width < destination data width => Right Aligned padded with sign extended up to destination data width

DMA_DATA_LEFTALIGN_RIGHTTRUNC

If source data width > destination data width => Left Aligned Right Truncated down to the destination data width

DMA_DATA_PACK

If source data width < destination data width => Packed at the destination data width (Not available on LPDMA)

DMA_DATA_UNPACK

If source data width > destination data width => Unpacked at the destination data width (Not available on LPDMA)

DMAEx Data Exchange

DMA_EXCHANGE_NONE

No data exchange

DMA_EXCHANGE_DEST_BYTE

Destination Byte exchange when destination data width is > Byte

DMA_EXCHANGE_DEST_HALFWORD

Destination Half-Word exchange when destination data width is > Half-Word

DMA_EXCHANGE_SRC_BYTE

Source Byte endianness exchange when source data width is word

DMAEx LinkedList Mode

DMA_LINKEDLIST_NORMAL

Linear linked-list DMA channel transfer

DMA_LINKEDLIST_CIRCULAR

Circular linked-list DMA channel transfer

DMAEx Linked-List Allocated Port

DMA_LINK_ALLOCATED_PORT0

Link allocated port 0

DMA_LINK_ALLOCATED_PORT1

Link allocated port 1

DMAEx Link Step Mode

DMA_LSM_FULL_EXECUTION

Channel is executed for the full linked-list

DMA_LSM_1LINK_EXECUTION

Channel is executed once for the current LLI

DMAEx Node Type

DMA_LPDMA_LINEAR_NODE

Defines the LPDMA linear addressing node type

DMA_GPDMA_LINEAR_NODE

Defines the GPDMA linear addressing node type

DMA_GPDMA_2D_NODE

Defines the GPDMA 2 dimension addressing node type

DMAEx Trigger Mode

DMA_TRIGM_BLOCK_TRANSFER

A block transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_REPEATED_BLOCK_TRANSFER

A repeated block transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_LLI_LINK_TRANSFER

A LLI link transfer is conditioned by (at least) one hit trigger

DMA_TRIGM_SINGLE_BURST_TRANSFER

A single/burst transfer is conditioned by (at least) one hit trigger

DMAEx Trigger Polarity

DMA_TRIG_POLARITY_MASKED

No trigger of the selected DMA request. Masked trigger event

DMA_TRIG_POLARITY_RISING

Trigger of the selected DMA request on the rising edge of the selected trigger event input

DMA_TRIG_POLARITY_FALLING

Trigger of the selected DMA request on the falling edge of the selected trigger event input

DMAEx Trigger Selection

GPDMA1_TRIGGER_EXTI_LINE0

GPDMA1 HW Trigger signal is EXTI_LINE0

GPDMA1_TRIGGER_EXTI_LINE1

GPDMA1 HW Trigger signal is EXTI_LINE1

GPDMA1_TRIGGER_EXTI_LINE2

GPDMA1 HW Trigger signal is EXTI_LINE2

GPDMA1_TRIGGER_EXTI_LINE3

GPDMA1 HW Trigger signal is EXTI_LINE3

GPDMA1_TRIGGER_EXTI_LINE4

GPDMA1 HW Trigger signal is EXTI_LINE4

GPDMA1_TRIGGER_EXTI_LINE5

GPDMA1 HW Trigger signal is EXTI_LINE5

GPDMA1_TRIGGER_EXTI_LINE6

GPDMA1 HW Trigger signal is EXTI_LINE6

GPDMA1_TRIGGER_EXTI_LINE7
GPDMA1 HW Trigger signal is EXTI_LINE7

GPDMA1_TRIGGER_TAMP_TRG1
GPDMA1 HW Trigger signal is TAMP_TRG1

GPDMA1_TRIGGER_TAMP_TRG2
GPDMA1 HW Trigger signal is TAMP_TRG2

GPDMA1_TRIGGER_TAMP_TRG3
GPDMA1 HW Trigger signal is TAMP_TRG3

GPDMA1_TRIGGER_LPTIM1_CH1
GPDMA1 HW Trigger signal is LPTIM1_CH1

GPDMA1_TRIGGER_LPTIM1_CH2
GPDMA1 HW Trigger signal is LPTIM1_CH2

GPDMA1_TRIGGER_LPTIM2_CH1
GPDMA1 HW Trigger signal is LPTIM2_CH1

GPDMA1_TRIGGER_LPTIM2_CH2
GPDMA1 HW Trigger signal is LPTIM2_CH2

GPDMA1_TRIGGER_LPTIM4_OUT
GPDMA1 HW Trigger signal is LPTIM4_OUT

GPDMA1_TRIGGER_COMP1_OUT
GPDMA1 HW Trigger signal is COMP1_OUT

GPDMA1_TRIGGER_COMP2_OUT
GPDMA1 HW Trigger signal is COMP2_OUT

GPDMA1_TRIGGER_RTC_ALRA_TRG
GPDMA1 HW Trigger signal is RTC_ALRA_TRG

GPDMA1_TRIGGER_RTC_ALRB_TRG
GPDMA1 HW Trigger signal is RTC_ALRB_TRG

GPDMA1_TRIGGER_RTC_WUT_TRG
GPDMA1 HW Trigger signal is RTC_WUT_TRG

GPDMA1_TRIGGER_GPDMA1_CH0_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

GPDMA1_TRIGGER_GPDMA1_CH1_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

GPDMA1_TRIGGER_GPDMA1_CH2_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH2_TCF

GPDMA1_TRIGGER_GPDMA1_CH3_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH3_TCF

GPDMA1_TRIGGER_GPDMA1_CH4_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

GPDMA1_TRIGGER_GPDMA1_CH5_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

GPDMA1_TRIGGER_GPDMA1_CH6_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

GPDMA1_TRIGGER_GPDMA1_CH7_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

GPDMA1_TRIGGER_GPDMA1_CH8_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH8_TCF

GPDMA1_TRIGGER_GPDMA1_CH9_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH9_TCF

GPDMA1_TRIGGER_GPDMA1_CH10_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH10_TCF

GPDMA1_TRIGGER_GPDMA1_CH11_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH11_TCF

GPDMA1_TRIGGER_GPDMA1_CH12_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH12_TCF

GPDMA1_TRIGGER_GPDMA1_CH13_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH13_TCF

GPDMA1_TRIGGER_GPDMA1_CH14_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH14_TCF

GPDMA1_TRIGGER_GPDMA1_CH15_TCF
GPDMA1 HW Trigger signal is GPDMA1_CH15_TCF

GPDMA1_TRIGGER_LPDMA1_CH0_TCF
GPDMA1 HW Trigger signal is LPDMA1_CH0_TCF

GPDMA1_TRIGGER_LPDMA1_CH1_TCF
GPDMA1 HW Trigger signal is LPDMA1_CH1_TCF

GPDMA1_TRIGGER_LPDMA1_CH2_TCF
GPDMA1 HW Trigger signal is LPDMA1_CH2_TCF

GPDMA1_TRIGGER_LPDMA1_CH3_TCF
GPDMA1 HW Trigger signal is LPDMA1_CH3_TCF

GPDMA1_TRIGGER_TIM2_TRGO
GPDMA1 HW Trigger signal is TIM2_TRGO

GPDMA1_TRIGGER_TIM15_TRGO
GPDMA1 HW Trigger signal is TIM15_TRGO

GPDMA1_TRIGGER_TIM3_TRGO
GPDMA1 HW Trigger signal is TIM3_TRGO

GPDMA1_TRIGGER_TIM4_TRGO
GPDMA1 HW Trigger signal is TIM4_TRGO

GPDMA1_TRIGGER_TIM5_TRGO
GPDMA1 HW Trigger signal is TIM5_TRGO

GPDMA1_TRIGGER_LTDC_LI
GPDMA1 HW Trigger signal is LTDC_LI

GPDMA1_TRIGGER_DSI_TE
GPDMA1 HW Trigger signal is DSI_TE

GPDMA1_TRIGGER_DSI_ER
GPDMA1 HW Trigger signal is DSI_ER

GPDMA1_TRIGGER_DMA2D_TC
GPDMA1 HW Trigger signal is DMA2D_TC

GPDMA1_TRIGGER_DMA2D_CTC
GPDMA1 HW Trigger signal is DMA2D_CTC

GPDMA1_TRIGGER_DMA2D_TW
GPDMA1 HW Trigger signal is DMA2D_TW

GPDMA1_TRIGGER_GPU2D_FLAG0
GPDMA1 HW Trigger signal is GPU2D_FLAG0

GPDMA1_TRIGGER_GPU2D_FLAG1
GPDMA1 HW Trigger signal is GPU2D_FLAG1

GPDMA1_TRIGGER_GPU2D_FLAG2
GPDMA1 HW Trigger signal is GPU2D_FLAG2

GPDMA1_TRIGGER_GPU2D_FLAG3
GPDMA1 HW Trigger signal is GPU2D_FLAG3

GPDMA1_TRIGGER_ADC4_AWD1
GPDMA1 HW Trigger signal is ADC4_AWD1

GPDMA1_TRIGGER_ADC1_AWD1
GPDMA1 HW Trigger signal is ADC1_AWD1

GPDMA1_TRIGGER_GFXTIM_EVT3
GPDMA1 HW Trigger signal is GFXTIM_EVT3

GPDMA1_TRIGGER_GFXTIM_EVT2
GPDMA1 HW Trigger signal is GFXTIM_EVT2

GPDMA1_TRIGGER_GFXTIM_EVT1
GPDMA1 HW Trigger signal is GFXTIM_EVT1

GPDMA1_TRIGGER_GFXTIM_EVT0
GPDMA1 HW Trigger signal is GFXTIM_EVT0

GPDMA1_TRIGGER_JPEG_EOC
GPDMA1 HW Trigger signal is JPEG_EOC

GPDMA1_TRIGGER_JPEG_IFNF
GPDMA1 HW Trigger signal is JPEG_IFNF

GPDMA1_TRIGGER_JPEG_IFT

GPDMA1 HW Trigger signal is JPEG_IFT

GPDMA1_TRIGGER_JPEG_OFNE

GPDMA1 HW Trigger signal is JPEG_OFNE

GPDMA1_TRIGGER_JPEG_OFT

GPDMA1 HW Trigger signal is JPEG_OFT

LPDMA1_TRIGGER_EXTI_LINE0

LPDMA1 HW Trigger signal is EXTI_LINE0

LPDMA1_TRIGGER_EXTI_LINE1

LPDMA1 HW Trigger signal is EXTI_LINE1

LPDMA1_TRIGGER_EXTI_LINE2

LPDMA1 HW Trigger signal is EXTI_LINE2

LPDMA1_TRIGGER_EXTI_LINE3

LPDMA1 HW Trigger signal is EXTI_LINE3

LPDMA1_TRIGGER_EXTI_LINE4

LPDMA1 HW Trigger signal is EXTI_LINE4

LPDMA1_TRIGGER_TAMP_TRG1

LPDMA1 HW Trigger signal is TAMP_TRG1

LPDMA1_TRIGGER_TAMP_TRG2

LPDMA1 HW Trigger signal is TAMP_TRG2

LPDMA1_TRIGGER_TAMP_TRG3

LPDMA1 HW Trigger signal is TAMP_TRG3

LPDMA1_TRIGGER_LPTIM1_CH1

LPDMA1 HW Trigger signal is LPTIM1_CH1

LPDMA1_TRIGGER_LPTIM1_CH2

LPDMA1 HW Trigger signal is LPTIM1_CH2

LPDMA1_TRIGGER_LPTIM3_CH1

LPDMA1 HW Trigger signal is LPTIM3_CH1

LPDMA1_TRIGGER_LPTIM4_OUT

LPDMA1 HW Trigger signal is LPTIM4_OUT

LPDMA1_TRIGGER_COMP1_OUT

LPDMA1 HW Trigger signal is COMP1_OUT

LPDMA1_TRIGGER_COMP2_OUT

LPDMA1 HW Trigger signal is COMP2_OUT

LPDMA1_TRIGGER_RTC_ALRA_TRG

LPDMA1 HW Trigger signal is RTC_ALRA_TRG

LPDMA1_TRIGGER_RTC_ALRB_TRG

LPDMA1 HW Trigger signal is RTC_ALRB_TRG

LPDMA1_TRIGGER_RTC_WUT_TRG
LPDMA1 HW Trigger signal is RTC_WUT_TRG

LPDMA1_TRIGGER_ADC4_AWD1
LPDMA1 HW Trigger signal is ADC4_AWD1

LPDMA1_TRIGGER_LPDMA1_CH0_TCF
LPDMA1 HW Trigger signal is LPDMA1_CH0_TCF

LPDMA1_TRIGGER_LPDMA1_CH1_TCF
LPDMA1 HW Trigger signal is LPDMA1_CH1_TCF

LPDMA1_TRIGGER_LPDMA1_CH2_TCF
LPDMA1 HW Trigger signal is LPDMA1_CH2_TCF

LPDMA1_TRIGGER_LPDMA1_CH3_TCF
LPDMA1 HW Trigger signal is LPDMA1_CH3_TCF

LPDMA1_TRIGGER_GPDMA1_CH0_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH0_TCF

LPDMA1_TRIGGER_GPDMA1_CH1_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH1_TCF

LPDMA1_TRIGGER_GPDMA1_CH4_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH4_TCF

LPDMA1_TRIGGER_GPDMA1_CH5_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH5_TCF

LPDMA1_TRIGGER_GPDMA1_CH6_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH6_TCF

LPDMA1_TRIGGER_GPDMA1_CH7_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH7_TCF

LPDMA1_TRIGGER_GPDMA1_CH12_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH12_TCF

LPDMA1_TRIGGER_GPDMA1_CH13_TCF
LPDMA1 HW Trigger signal is GPDMA1_CH13_TCF

LPDMA1_TRIGGER_TIM2_TRGO
LPDMA1 HW Trigger signal is TIM2_TRGO

LPDMA1_TRIGGER_TIM15_TRGO
LPDMA1 HW Trigger signal is TIM15_TRGO

23 HAL DSI Generic Driver

23.1 DSI Firmware driver registers structures

23.1.1 DSI_InitTypeDef

DSI_InitTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***uint32_t AutomaticClockLaneControl***
- ***uint32_t TXEscapeCkdiv***
- ***uint32_t NumberOfLanes***
- ***uint32_t PHYFrequencyRange***
- ***uint32_t PHYLowPowerOffset***

Field Documentation

- ***uint32_t DSI_InitTypeDef::AutomaticClockLaneControl***
  Automatic clock lane control This parameter can be any value of ***DSI_Automatic_Clk_Lane_Control***
- ***uint32_t DSI_InitTypeDef::TXEscapeCkdiv***
  TX Escape clock division The values 0 and 1 stop the TX_ESC clock generation
- ***uint32_t DSI_InitTypeDef::NumberOfLanes***
  Number of lanes This parameter can be any value of ***DSI_Number_Of_Lanes***
- ***uint32_t DSI_InitTypeDef::PHYFrequencyRange***
  PHY Frequency Range This parameter can be any value of
  - DSI_PHYFrequencyRange
- ***uint32_t DSI_InitTypeDef::PHYLowPowerOffset***
  PHY Low Power Offset This parameter can be any value of
  - PHYLowPowerOffset

23.1.2 DSI_PLLInitTypeDef

DSI_PLLInitTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***uint32_t PLLNDIV***
- ***uint32_t PLLIDF***
- ***uint32_t PLLODF***
- ***uint32_t PLLVCORange***
- ***uint32_t PLLChargePump***
- ***uint32_t PLLTuning***

Field Documentation

- ***uint32_t DSI_PLLInitTypeDef::PLLNDIV***
  PLL Loop Division Factor This parameter must be a value between 10 and 125
- ***uint32_t DSI_PLLInitTypeDef::PLLIDF***
  PLL Input Division Factor This parameter can be any value of ***DSI_PLL_IDF***
- ***uint32_t DSI_PLLInitTypeDef::PLLODF***
  PLL Output Division Factor This parameter can be any value of ***DSI_PLL_ODF***
- ***uint32_t DSI_PLLInitTypeDef::PLLVCORange***
  PLLVCORange This parameter can be any value of ***DSI_DPHY_VCO_Frequency_RANGE***
- ***uint32_t DSI_PLLInitTypeDef::PLLChargePump***
  PLLChargePump This parameter can be any value of ***DSI_PLL_Charge_Pump***
- ***uint32_t DSI_PLLInitTypeDef::PLLTuning***
  PLLTuning This parameter can be any value of ***DSI_PLL_Tuning***

23.1.3 DSI_VidCfgTypeDef

DSI_VidCfgTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- *uint32_t VirtualChannelID*
- *uint32_t ColorCoding*
- *uint32_t LooselyPacked*
- *uint32_t Mode*
- *uint32_t PacketSize*
- *uint32_t NumberOfChunks*
- *uint32_t NullPacketSize*
- *uint32_t HSPolarity*
- *uint32_t VSPolarity*
- *uint32_t DEPolarity*
- *uint32_t HorizontalSyncActive*
- *uint32_t HorizontalBackPorch*
- *uint32_t HorizontalLine*
- *uint32_t VerticalSyncActive*
- *uint32_t VerticalBackPorch*
- *uint32_t VerticalFrontPorch*
- *uint32_t VerticalActive*
- *uint32_t LPCommandEnable*
- *uint32_t LPLargestPacketSize*
- *uint32_t LPVACTLargestPacketSize*
- *uint32_t LPHorizontalFrontPorchEnable*
- *uint32_t LPHorizontalBackPorchEnable*
- *uint32_t LPVerticalActiveEnable*
- *uint32_t LPVerticalFrontPorchEnable*
- *uint32_t LPVerticalBackPorchEnable*
- *uint32_t LPVerticalSyncActiveEnable*
- *uint32_t FrameBTAAcknowledgeEnable*

Field Documentation

- ***uint32_t DSI_VidCfgTypeDef::VirtualChannelID***
  Virtual channel ID
- ***uint32_t DSI_VidCfgTypeDef::ColorCoding***
  Color coding for LTDC interface This parameter can be any value of *DSI_Color_Coding*
- ***uint32_t DSI_VidCfgTypeDef::LooselyPacked***
  Enable or disable loosely packed stream (needed only when using 18-bit configuration). This parameter can be any value of *DSI_LooselyPacked*
- ***uint32_t DSI_VidCfgTypeDef::Mode***
  Video mode type This parameter can be any value of *DSI_Video_Mode_Type*
- ***uint32_t DSI_VidCfgTypeDef::PacketSize***
  Video packet size
- ***uint32_t DSI_VidCfgTypeDef::NumberOfChunks***
  Number of chunks
- ***uint32_t DSI_VidCfgTypeDef::NullPacketSize***
  Null packet size
- ***uint32_t DSI_VidCfgTypeDef::HSPolarity***
  HSYNC pin polarity This parameter can be any value of *DSI_HSYNC_Polarity*
- ***uint32_t DSI_VidCfgTypeDef::VSPolarity***
  VSYNC pin polarity This parameter can be any value of *DSI_VSYNC_Active_Polarity*
- ***uint32_t DSI_VidCfgTypeDef::DEPolarity***
  Data Enable pin polarity This parameter can be any value of *DSI_DATA_ENABLE_Polarity*

- ***uint32_t DSI_VidCfgTypeDef::HorizontalSyncActive***
  Horizontal synchronism active duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::HorizontalBackPorch***
  Horizontal back-porch duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::HorizontalLine***
  Horizontal line duration (in lane byte clock cycles)
- ***uint32_t DSI_VidCfgTypeDef::VerticalSyncActive***
  Vertical synchronism active duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalBackPorch***
  Vertical back-porch duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalFrontPorch***
  Vertical front-porch duration
- ***uint32_t DSI_VidCfgTypeDef::VerticalActive***
  Vertical active duration
- ***uint32_t DSI_VidCfgTypeDef::LPCommandEnable***
  Low-power command enable This parameter can be any value of ***DSI_LP_Command***
- ***uint32_t DSI_VidCfgTypeDef::LPLargestPacketSize***
  The size, in bytes, of the low power largest packet that can fit in a line during VSA, VBP and VFP regions
- ***uint32_t DSI_VidCfgTypeDef::LPVACTLargestPacketSize***
  The size, in bytes, of the low power largest packet that can fit in a line during VACT region
- ***uint32_t DSI_VidCfgTypeDef::LPHorizontalFrontPorchEnable***
  Low-power horizontal front-porch enable This parameter can be any value of ***DSI_LP_HFP***
- ***uint32_t DSI_VidCfgTypeDef::LPHorizontalBackPorchEnable***
  Low-power horizontal back-porch enable This parameter can be any value of ***DSI_LP_HBP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalActiveEnable***
  Low-power vertical active enable This parameter can be any value of ***DSI_LP_VACT***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalFrontPorchEnable***
  Low-power vertical front-porch enable This parameter can be any value of ***DSI_LP_VFP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalBackPorchEnable***
  Low-power vertical back-porch enable This parameter can be any value of ***DSI_LP_VBP***
- ***uint32_t DSI_VidCfgTypeDef::LPVerticalSyncActiveEnable***
  Low-power vertical sync active enable This parameter can be any value of ***DSI_LP_VSYNC***
- ***uint32_t DSI_VidCfgTypeDef::FrameBTAAcknowledgeEnable***
  Frame bus-turn-around acknowledge enable This parameter can be any value of ***DSI_FBTA_acknowledge***

23.1.4 DSI_CmdCfgTypeDef

DSI_CmdCfgTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***uint32_t VirtualChannelID***
- ***uint32_t ColorCoding***
- ***uint32_t CommandSize***
- ***uint32_t TearingEffectSource***
- ***uint32_t TearingEffectPolarity***
- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t VSyncPol***
- ***uint32_t AutomaticRefresh***
- ***uint32_t TEAcknowledgeRequest***

Field Documentation

- ***uint32_t DSI_CmdCfgTypeDef::VirtualChannelID***
  Virtual channel ID

- ***uint32_t DSI_CmdCfgTypeDef::ColorCoding***
  Color coding for LTDC interface This parameter can be any value of ***DSI_Color_Coding***
- ***uint32_t DSI_CmdCfgTypeDef::CommandSize***
  Maximum allowed size for an LTDC write memory command, measured in pixels. This parameter can be any value between 0x00 and 0xFFFFU
- ***uint32_t DSI_CmdCfgTypeDef::TearingEffectSource***
  Tearing effect source This parameter can be any value of ***DSI_TearingEffectSource***
- ***uint32_t DSI_CmdCfgTypeDef::TearingEffectPolarity***
  Tearing effect pin polarity This parameter can be any value of ***DSI_TearingEffectPolarity***
- ***uint32_t DSI_CmdCfgTypeDef::HSPolarity***
  HSYNC pin polarity This parameter can be any value of ***DSI_HSYNC_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::VSPolarity***
  VSYNC pin polarity This parameter can be any value of ***DSI_VSYNC_Active_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::DEPolarity***
  Data Enable pin polarity This parameter can be any value of ***DSI_DATA_ENABLE_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::VSyncPol***
  VSync edge on which the LTDC is halted This parameter can be any value of ***DSI_Vsync_Polarity***
- ***uint32_t DSI_CmdCfgTypeDef::AutomaticRefresh***
  Automatic refresh mode This parameter can be any value of ***DSI_AutomaticRefresh***
- ***uint32_t DSI_CmdCfgTypeDef::TEAcknowledgeRequest***
  Tearing Effect Acknowledge Request Enable This parameter can be any value of ***DSI_TE_AcknowledgeRequest***

23.1.5 DSI_LPCmdTypeDef

DSI_LPCmdTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***uint32_t LPGenShortWriteNoP***
- ***uint32_t LPGenShortWriteOneP***
- ***uint32_t LPGenShortWriteTwoP***
- ***uint32_t LPGenShortReadNoP***
- ***uint32_t LPGenShortReadOneP***
- ***uint32_t LPGenShortReadTwoP***
- ***uint32_t LPGenLongWrite***
- ***uint32_t LPDcsShortWriteNoP***
- ***uint32_t LPDcsShortWriteOneP***
- ***uint32_t LPDcsShortReadNoP***
- ***uint32_t LPDcsLongWrite***
- ***uint32_t LPMaxReadPacket***
- ***uint32_t AcknowledgeRequest***

Field Documentation

- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteNoP***
  Generic Short Write Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteOneP***
  Generic Short Write One parameter Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortWriteTwoP***
  Generic Short Write Two parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortWriteTwoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadNoP***
  Generic Short Read Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadNoP***

- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadOneP***
  Generic Short Read One parameter Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenShortReadTwoP***
  Generic Short Read Two parameters Transmission This parameter can be any value of ***DSI_LP_LPGenShortReadTwoP***
- ***uint32_t DSI_LPCmdTypeDef::LPGenLongWrite***
  Generic Long Write Transmission This parameter can be any value of ***DSI_LP_LPGenLongWrite***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteNoP***
  DCS Short Write Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPDcsShortWriteNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortWriteOneP***
  DCS Short Write One parameter Transmission This parameter can be any value of ***DSI_LP_LPDcsShortWriteOneP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsShortReadNoP***
  DCS Short Read Zero parameters Transmission This parameter can be any value of ***DSI_LP_LPDcsShortReadNoP***
- ***uint32_t DSI_LPCmdTypeDef::LPDcsLongWrite***
  DCS Long Write Transmission This parameter can be any value of ***DSI_LP_LPDcsLongWrite***
- ***uint32_t DSI_LPCmdTypeDef::LPMaxReadPacket***
  Maximum Read Packet Size Transmission This parameter can be any value of ***DSI_LP_LPMaxReadPacket***
- ***uint32_t DSI_LPCmdTypeDef::AcknowledgeRequest***
  Acknowledge Request Enable This parameter can be any value of ***DSI_AcknowledgeRequest***

23.1.6 DSI_PHY_TimerTypeDef

DSI_PHY_TimerTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***uint32_t ClockLaneHS2LPTime***
- ***uint32_t ClockLaneLP2HSTime***
- ***uint32_t DataLaneHS2LPTime***
- ***uint32_t DataLaneLP2HSTime***
- ***uint32_t DataLaneMaxReadTime***
- ***uint32_t StopWaitTime***

Field Documentation

- ***uint32_t DSI_PHY_TimerTypeDef::ClockLaneHS2LPTime***
  The maximum time that the D-PHY clock lane takes to go from high-speed to low-power transmission
- ***uint32_t DSI_PHY_TimerTypeDef::ClockLaneLP2HSTime***
  The maximum time that the D-PHY clock lane takes to go from low-power to high-speed transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneHS2LPTime***
  The maximum time that the D-PHY data lanes takes to go from high-speed to low-power transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneLP2HSTime***
  The maximum time that the D-PHY data lanes takes to go from low-power to high-speed transmission
- ***uint32_t DSI_PHY_TimerTypeDef::DataLaneMaxReadTime***
  The maximum time required to perform a read command
- ***uint32_t DSI_PHY_TimerTypeDef::StopWaitTime***
  The minimum wait period to request a High-Speed transmission after the Stop state

23.1.7 DSI_HOST_TimeoutTypeDef

DSI_HOST_TimeoutTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***uint32_t TimeoutCkdiv***
- ***uint32_t HighSpeedTransmissionTimeout***
- ***uint32_t LowPowerReceptionTimeout***

- *uint32_t HighSpeedReadTimeout*
- *uint32_t LowPowerReadTimeout*
- *uint32_t HighSpeedWriteTimeout*
- *uint32_t HighSpeedWritePrespMode*
- *uint32_t LowPowerWriteTimeout*
- *uint32_t BTATimeout*

Field Documentation

- ***uint32_t DSI_HOST_TimeoutTypeDef::TimeoutCkdiv***
  Time-out clock division
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedTransmissionTimeout***
  High-speed transmission time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerReceptionTimeout***
  Low-power reception time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedReadTimeout***
  High-speed read time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerReadTimeout***
  Low-power read time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedWriteTimeout***
  High-speed write time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::HighSpeedWritePrespMode***
  High-speed write presp mode This parameter can be any value of ***DSI_HS_PrespMode***
- ***uint32_t DSI_HOST_TimeoutTypeDef::LowPowerWriteTimeout***
  Low-speed write time-out
- ***uint32_t DSI_HOST_TimeoutTypeDef::BTATimeout***
  BTA time-out

23.1.8 __DSI_HandleTypeDef

__DSI_HandleTypeDef is defined in the stm32u5xx_hal_dsi.h

Data Fields

- ***DSI_TypeDef * Instance***
- ***DSI_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t ErrorMsk***
- ***void(* TearingEffectCallback***
- ***void(* EndOfRefreshCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***DSI_TypeDef* __DSI_HandleTypeDef::Instance***
  Register base address
- ***DSI_InitTypeDef __DSI_HandleTypeDef::Init***
  DSI required parameters
- ***HAL_LockTypeDef __DSI_HandleTypeDef::Lock***
  DSI peripheral status
- ***__IO HAL_DSI_StateTypeDef __DSI_HandleTypeDef::State***
  DSI communication state
- ***__IO uint32_t __DSI_HandleTypeDef::ErrorCode***
  DSI Error code
- ***uint32_t __DSI_HandleTypeDef::ErrorMsk***
  DSI Error monitoring mask

- ***void(* __DSI_HandleTypeDef::TearingEffectCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Tearing Effect Callback
- ***void(* __DSI_HandleTypeDef::EndOfRefreshCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI End Of Refresh Callback
- ***void(* __DSI_HandleTypeDef::ErrorCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Error Callback
- ***void(* __DSI_HandleTypeDef::MspInitCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Msp Init callback
- ***void(* __DSI_HandleTypeDef::MspDeInitCallback)(struct __DSI_HandleTypeDef *hdsi)***
  DSI Msp DeInit callback

23.2 DSI Firmware driver API description

The following section lists the various functions of the DSI library.

23.2.1 How to use this driver

The DSI HAL driver can be used as follows:

1. Declare a DSI_HandleTypeDef handle structure, for example: DSI_HandleTypeDef hdsi;
2. Initialize the DSI low level resources by implementing the HAL_DSI_MspInit() API:
   a. Enable the DSI interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the DSI interrupt priority
      - Enable the NVIC DSI IRQ Channel
3. Initialize the DSI Host peripheral, the required PLL parameters, number of lances and TX Escape clock divider by calling the HAL_DSI_Init() API which calls HAL_DSI_MspInit().

Configuration

1. Use HAL_DSI_ConfigAdaptedCommandMode() function to configure the DSI host in adapted command mode.
2. When operating in video mode , use HAL_DSI_ConfigVideoMode() to configure the DSI host.
3. Function HAL_DSI_ConfigCommand() is used to configure the DSI commands behavior in low power mode.
4. To configure the DSI PHY timings parameters, use function HAL_DSI_ConfigPhyTimer().
5. The DSI Host can be started/stopped using respectively functions HAL_DSI_Start() and HAL_DSI_Stop(). Functions HAL_DSI_ShortWrite(), HAL_DSI_LongWrite() and HAL_DSI_Read() allows respectively to write DSI short packets, long packets and to read DSI packets.
6. The DSI Host Offers two Low power modes :
   – Low Power Mode on data lanes only: Only DSI data lanes are shut down. It is possible to enter/exit from this mode using respectively functions HAL_DSI_EnterULPMData() and HAL_DSI_ExitULPMData()
   – Low Power Mode on data and clock lanes : All DSI lanes are shut down including data and clock lanes. It is possible to enter/exit from this mode using respectively functions HAL_DSI_EnterULPM() and HAL_DSI_ExitULPM()
7. To control DSI state you can use the following function: HAL_DSI_GetState()

Error management

1. User can select the DSI errors to be reported/monitored using function HAL_DSI_ConfigErrorMonitor() When an error occurs, the callback HAL_DSI_ErrorCallback() is asserted and then user can retrieve the error code by calling function HAL_DSI_GetError()

DSI HAL driver macros list

Below the list of most used macros in DSI HAL driver.

- __HAL_DSI_ENABLE: Enable the DSI Host.
- __HAL_DSI_DISABLE: Disable the DSI Host.
- __HAL_DSI_WRAPPER_ENABLE: Enables the DSI wrapper.

- __HAL_DSI_WRAPPER_DISABLE: Disable the DSI wrapper.
- __HAL_DSI_PLL_ENABLE: Enables the DSI PLL.
- __HAL_DSI_PLL_DISABLE: Disables the DSI PLL.
- __HAL_DSI_REG_ENABLE: Enables the DSI regulator.
- __HAL_DSI_REG_DISABLE: Disables the DSI regulator.
- __HAL_DSI_GET_FLAG: Get the DSI pending flags.
- __HAL_DSI_CLEAR_FLAG: Clears the DSI pending flags.
- __HAL_DSI_ENABLE_IT: Enables the specified DSI interrupts.
- __HAL_DSI_DISABLE_IT: Disables the specified DSI interrupts.
- __HAL_DSI_GET_IT_SOURCE: Checks whether the specified DSI interrupt source is enabled or not.

Note: You can refer to the DSI HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_DSI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Function HAL_DSI_RegisterCallback() to register a callback.

Function HAL_DSI_RegisterCallback() allows to register following callbacks:

- TearingEffectCallback : DSI Tearing Effect Callback.
- EndOfRefreshCallback : DSI End Of Refresh Callback.
- ErrorCallback : DSI Error Callback
- MspInitCallback : DSI MspInit.
- MspDeInitCallback : DSI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_DSI_UnRegisterCallback() to reset a callback to the default weak function. HAL_DSI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- TearingEffectCallback : DSI Tearing Effect Callback.
- EndOfRefreshCallback : DSI End Of Refresh Callback.
- ErrorCallback : DSI Error Callback
- MspInitCallback : DSI MspInit.
- MspDeInitCallback : DSI MspDeInit.

By default, after the HAL_DSI_Init and when the state is HAL_DSI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_DSI_TearingEffectCallback(), HAL_DSI_EndOfRefreshCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (overridden) functions in the HAL_DSI_Init() and HAL_DSI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_DSI_Init() and HAL_DSI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_DSI_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_DSI_STATE_READY or HAL_DSI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_DSI_RegisterCallback() before calling HAL_DSI_DeInit() or HAL_DSI_Init() function.

When The compilation define USE_HAL_DSI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

23.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the DSI
- De-initialize the DSI

This section contains the following APIs:

- *HAL_DSI_Init()*
- *HAL_DSI_DeInit()*

- *HAL_DSI_ConfigErrorMonitor()*
- *HAL_DSI_MspInit()*
- *HAL_DSI_MspDeInit()*
- *HAL_DSI_RegisterCallback()*
- *HAL_DSI_UnRegisterCallback()*

23.2.3 IO operation functions

This section provides function allowing to:

- Handle DSI interrupt request

This section contains the following APIs:

- *HAL_DSI_IRQHandler()*
- *HAL_DSI_TearingEffectCallback()*
- *HAL_DSI_EndOfRefreshCallback()*
- *HAL_DSI_ErrorCallback()*

23.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the Generic interface read-back Virtual Channel ID
- Select video mode and configure the corresponding parameters
- Configure command transmission mode: High-speed or Low-power
- Configure the flow control
- Configure the DSI PHY timer
- Configure the DSI HOST timeout
- Configure the DSI HOST timeout
- Start/Stop the DSI module
- Refresh the display in command mode
- Controls the display color mode in Video mode
- Control the display shutdown in Video mode
- write short DCS or short Generic command
- write long DCS or long Generic command
- Read command (DCS or generic)
- Enter/Exit the Ultra Low Power Mode on data only (D-PHY PLL running)
- Enter/Exit the Ultra Low Power Mode on data only and clock (D-PHY PLL turned off)
- Start/Stop test pattern generation
- Slew-Rate And Delay Tuning
- Low-Power Reception Filter Tuning
- Activate an additional current path on all lanes to meet the SDDTx parameter
- Custom lane pins configuration
- Set custom timing for the PHY
- Force the Clock/Data Lane in TX Stop Mode
- Force LP Receiver in Low-Power Mode
- Force Data Lanes in RX Mode after a BTA
- Enable a pull-down on the lanes to prevent from floating states when unused
- Switch off the contention detection on data lanes

This section contains the following APIs:

- *HAL_DSI_SetGenericVCIDRX()*
- *HAL_DSI_SetGenericVCIDTX()*
- *HAL_DSI_SetGenericVCID()*
- *HAL_DSI_ConfigVideoMode()*

- *HAL_DSI_ConfigAdaptedCommandMode()*
- *HAL_DSI_ConfigCommand()*
- *HAL_DSI_ConfigFlowControl()*
- *HAL_DSI_ConfigPhyTimer()*
- *HAL_DSI_ConfigHostTimeouts()*
- *HAL_DSI_Start()*
- *HAL_DSI_Stop()*
- *HAL_DSI_Refresh()*
- *HAL_DSI_ColorMode()*
- *HAL_DSI_Shutdown()*
- *HAL_DSI_ShortWrite()*
- *HAL_DSI_LongWrite()*
- *HAL_DSI_Read()*
- *HAL_DSI_EnterULPMData()*
- *HAL_DSI_ExitULPMData()*
- *HAL_DSI_EnterULPM()*
- *HAL_DSI_ExitULPM()*
- *HAL_DSI_PatternGeneratorStart()*
- *HAL_DSI_PatternGeneratorStop()*
- *HAL_DSI_SetLanePinsConfiguration()*
- *HAL_DSI_ForceTXStopMode()*

23.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to
- Check the DSI state.
- Get error code.

This section contains the following APIs:
- *HAL_DSI_GetState()*
- *HAL_DSI_GetError()*

23.2.6 Detailed description of functions

HAL_DSI_Init

Function name

HAL_StatusTypeDef HAL_DSI_Init (DSI_HandleTypeDef * hdsi, DSI_PLLInitTypeDef * PLLInit)

Function description

Initializes the DSI according to the specified parameters in the DSI_InitTypeDef and create the associated handle.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **PLLInit:** pointer to a DSI_PLLInitTypeDef structure that contains the PLL Clock structure definition for the DSI.

Return values

- **HAL:** status

HAL_DSI_DeInit

Function name

HAL_StatusTypeDef HAL_DSI_DeInit (DSI_HandleTypeDef * hdsi)

Function description

De-initializes the DSI peripheral registers to their default reset values.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_MspInit

Function name

void HAL_DSI_MspInit (DSI_HandleTypeDef * hdsi)

Function description

Initializes the DSI MSP.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_MspDeInit

Function name

void HAL_DSI_MspDeInit (DSI_HandleTypeDef * hdsi)

Function description

De-initializes the DSI MSP.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_ConfigErrorMonitor

Function name

HAL_StatusTypeDef HAL_DSI_ConfigErrorMonitor (DSI_HandleTypeDef * hdsi, uint32_t ActiveErrors)

Function description

Enable the error monitor flags.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ActiveErrors:** indicates which error interrupts will be enabled. This parameter can be any combination of
  - DSI_Error_Data_Type.

Return values

- **HAL:** status

HAL_DSI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_DSI_RegisterCallback (DSI_HandleTypeDef * hdsi, HAL_DSI_CallbackIDTypeDef CallbackID, pDSI_CallbackTypeDef pCallback)

Function description

Register a User DSI Callback To be used instead of the weak predefined callback.

Parameters

- **hdsi:** dsi handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_DSI_TEARING_EFFECT_CB_ID Tearing Effect Callback ID
  – HAL_DSI_ENDOF_REFRESH_CB_ID End Of Refresh Callback ID
  – HAL_DSI_ERROR_CB_ID Error Callback ID
  – HAL_DSI_MSPINIT_CB_ID MspInit callback ID
  – HAL_DSI_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_DSI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DSI_UnRegisterCallback (DSI_HandleTypeDef * hdsi, HAL_DSI_CallbackIDTypeDef CallbackID)

Function description

Unregister a DSI Callback DSI callback is redirected to the weak predefined callback.

Parameters

- **hdsi:** dsi handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_DSI_TEARING_EFFECT_CB_ID Tearing Effect Callback ID
  – HAL_DSI_ENDOF_REFRESH_CB_ID End Of Refresh Callback ID
  – HAL_DSI_ERROR_CB_ID Error Callback ID
  – HAL_DSI_MSPINIT_CB_ID MspInit callback ID
  – HAL_DSI_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_DSI_IRQHandler

Function name

void HAL_DSI_IRQHandler (DSI_HandleTypeDef * hdsi)

Function description

Handles DSI interrupt request.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_TearingEffectCallback

Function name

void HAL_DSI_TearingEffectCallback (DSI_HandleTypeDef * hdsi)

Function description

Tearing Effect DSI callback.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_EndOfRefreshCallback

Function name

void HAL_DSI_EndOfRefreshCallback (DSI_HandleTypeDef * hdsi)

Function description

End of Refresh DSI callback.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_ErrorCallback

Function name

void HAL_DSI_ErrorCallback (DSI_HandleTypeDef * hdsi)

Function description

Operation Error DSI callback.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **None:**

HAL_DSI_SetGenericVCIDRX

Function name

HAL_StatusTypeDef HAL_DSI_SetGenericVCIDRX (DSI_HandleTypeDef * hdsi, uint32_t VirtualChannelIdRx)

Function description

Configure the Generic interface read-back Virtual Channel ID.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **VirtualChannelIdRx:** Virtual channel ID

Return values

- **HAL:** status

HAL_DSI_SetGenericVCIDTX

Function name

HAL_StatusTypeDef HAL_DSI_SetGenericVCIDTX (DSI_HandleTypeDef * hdsi, uint32_t VirtualChannelIdTx)

Function description

Configure the Virtual channel ID for transmission.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **VirtualChannelIdTx:** Virtual channel ID

Return values

- **HAL:** status

HAL_DSI_SetGenericVCID

Function name

HAL_StatusTypeDef HAL_DSI_SetGenericVCID (DSI_HandleTypeDef * hdsi, uint32_t VirtualChannelID)

Function description

Configure the Generic interface read-back Virtual Channel ID.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **VirtualChannelID:** Virtual channel ID

Return values

- **HAL:** status

HAL_DSI_ConfigVideoMode

Function name

HAL_StatusTypeDef HAL_DSI_ConfigVideoMode (DSI_HandleTypeDef * hdsi, DSI_VidCfgTypeDef * VidCfg)

Function description

Select video mode and configure the corresponding parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **VidCfg:** pointer to a DSI_VidCfgTypeDef structure that contains the DSI video mode configuration parameters

Return values

- **HAL:** status

HAL_DSI_ConfigAdaptedCommandMode

Function name

HAL_StatusTypeDef HAL_DSI_ConfigAdaptedCommandMode (DSI_HandleTypeDef * hdsi, DSI_CmdCfgTypeDef * CmdCfg)

Function description

Select adapted command mode and configure the corresponding parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **CmdCfg:** pointer to a DSI_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters

Return values

- **HAL:** status

HAL_DSI_ConfigCommand

Function name

HAL_StatusTypeDef HAL_DSI_ConfigCommand (DSI_HandleTypeDef * hdsi, DSI_LPCmdTypeDef * LPCmd)

Function description

Configure command transmission mode: High-speed or Low-power and enable/disable acknowledge request after packet transmission.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **LPCmd:** pointer to a DSI_LPCmdTypeDef structure that contains the DSI command transmission mode configuration parameters

Return values

- **HAL:** status

HAL_DSI_ConfigFlowControl

Function name

HAL_StatusTypeDef HAL_DSI_ConfigFlowControl (DSI_HandleTypeDef * hdsi, uint32_t FlowControl)

Function description

Configure the flow control parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **FlowControl:** flow control feature(s) to be enabled. This parameter can be any combination of
  - DSI_FlowControl.

Return values

- **HAL:** status

HAL_DSI_ConfigPhyTimer

Function name

HAL_StatusTypeDef HAL_DSI_ConfigPhyTimer (DSI_HandleTypeDef * hdsi, DSI_PHY_TimerTypeDef * PhyTimers)

Function description

Configure the DSI PHY timer parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **PhyTimers:** DSI_PHY_TimerTypeDef structure that contains the DSI PHY timing parameters

Return values

- **HAL:** status

HAL_DSI_ConfigHostTimeouts

Function name

HAL_StatusTypeDef HAL_DSI_ConfigHostTimeouts (DSI_HandleTypeDef * hdsi, DSI_HOST_TimeoutTypeDef * HostTimeouts)

Function description

Configure the DSI HOST timeout parameters.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **HostTimeouts:** DSI_HOST_TimeoutTypeDef structure that contains the DSI host timeout parameters

Return values

- **HAL:** status

HAL_DSI_Start

Function name

HAL_StatusTypeDef HAL_DSI_Start (DSI_HandleTypeDef * hdsi)

Function description

Start the DSI module.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_Stop

Function name

HAL_StatusTypeDef HAL_DSI_Stop (DSI_HandleTypeDef * hdsi)

Function description

Stop the DSI module.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_Refresh

Function name

HAL_StatusTypeDef HAL_DSI_Refresh (DSI_HandleTypeDef * hdsi)

Function description

Refresh the display in command mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_ColorMode

Function name

HAL_StatusTypeDef HAL_DSI_ColorMode (DSI_HandleTypeDef * hdsi, uint32_t ColorMode)

Function description

Controls the display color mode in Video mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ColorMode:** Color mode (full or 8-colors). This parameter can be any value of
  – DSI_Color_Mode

Return values

- **HAL:** status

HAL_DSI_Shutdown

Function name

HAL_StatusTypeDef HAL_DSI_Shutdown (DSI_HandleTypeDef * hdsi, uint32_t Shutdown)

Function description

Control the display shutdown in Video mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Shutdown:** Shut-down (Display-ON or Display-OFF). This parameter can be any value of
  – DSI_ShutDown

Return values

- **HAL:** status

HAL_DSI_ShortWrite

Function name

HAL_StatusTypeDef HAL_DSI_ShortWrite (DSI_HandleTypeDef * hdsi, uint32_t ChannelID, uint32_t Mode, uint32_t Param1, uint32_t Param2)

Function description

write short DCS or short Generic command

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelID:** Virtual channel ID.
- **Mode:** DSI short packet data type. This parameter can be any value of
  – DSI_SHORT_WRITE_PKT_Data_Type.
- **Param1:** DSC command or first generic parameter. This parameter can be any value of
  – DSI_DCS_Command or a generic command code.
- **Param2:** DSC parameter or second generic parameter.

Return values

- **HAL:** status

HAL_DSI_LongWrite

Function name

HAL_StatusTypeDef HAL_DSI_LongWrite (DSI_HandleTypeDef * hdsi, uint32_t ChannelID, uint32_t Mode, uint32_t NbParams, uint32_t Param1, uint8_t * ParametersTable)

Function description

write long DCS or long Generic command

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelID:** Virtual channel ID.
- **Mode:** DSI long packet data type. This parameter can be any value of
  – DSI_LONG_WRITE_PKT_Data_Type.
- **NbParams:** Number of parameters.
- **Param1:** DSC command or first generic parameter. This parameter can be any value of
  – DSI_DCS_Command or a generic command code
- **ParametersTable:** Pointer to parameter values table.

Return values

- **HAL:** status

HAL_DSI_Read

Function name

HAL_StatusTypeDef HAL_DSI_Read (DSI_HandleTypeDef * hdsi, uint32_t ChannelNbr, uint8_t * Array, uint32_t Size, uint32_t Mode, uint32_t DCSCmd, uint8_t * ParametersTable)

Function description

Read command (DCS or generic)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **ChannelNbr:** Virtual channel ID
- **Array:** pointer to a buffer to store the payload of a read back operation.
- **Size:** Data size to be read (in byte).
- **Mode:** DSI read packet data type. This parameter can be any value of
  – DSI_SHORT_READ_PKT_Data_Type.
- **DCSCmd:** DCS get/read command.
- **ParametersTable:** Pointer to parameter values table.

Return values

- **HAL:** status

HAL_DSI_EnterULPMData

Function name

HAL_StatusTypeDef HAL_DSI_EnterULPMData (DSI_HandleTypeDef * hdsi)

Function description

Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_ExitULPMData

Function name

HAL_StatusTypeDef HAL_DSI_ExitULPMData (DSI_HandleTypeDef * hdsi)

Function description

Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL running (only data lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_EnterULPM

Function name

HAL_StatusTypeDef HAL_DSI_EnterULPM (DSI_HandleTypeDef * hdsi)

Function description

Enter the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_ExitULPM

Function name

HAL_StatusTypeDef HAL_DSI_ExitULPM (DSI_HandleTypeDef * hdsi)

Function description

Exit the ULPM (Ultra Low Power Mode) with the D-PHY PLL turned off (both data and clock lanes are in ULPM)

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_PatternGeneratorStart

Function name

HAL_StatusTypeDef HAL_DSI_PatternGeneratorStart (DSI_HandleTypeDef * hdsi, uint32_t Mode, uint32_t Orientation)

Function description

Start test pattern generation.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Mode:** Pattern generator mode This parameter can be one of the following values: 0 : Color bars (horizontal or vertical) 1 : BER pattern (vertical only)
- **Orientation:** Pattern generator orientation This parameter can be one of the following values: 0 : Vertical color bars 1 : Horizontal color bars

Return values

- **HAL:** status

HAL_DSI_PatternGeneratorStop

Function name

HAL_StatusTypeDef HAL_DSI_PatternGeneratorStop (DSI_HandleTypeDef * hdsi)

Function description

Stop test pattern generation.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** status

HAL_DSI_SetLanePinsConfiguration

Function name

HAL_StatusTypeDef HAL_DSI_SetLanePinsConfiguration (DSI_HandleTypeDef * hdsi, uint32_t CustomLane, uint32_t Lane, FunctionalState State)

Function description

Custom lane pins configuration.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **CustomLane:** Function to be applied on selected lane. This parameter can be any value of
  – DSI_CustomLane
- **Lane:** select between clock or data lane 0 or data lane 1. This parameter can be any value of
  – DSI_Lane_Select
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_ForceTXStopMode

Function name

HAL_StatusTypeDef HAL_DSI_ForceTXStopMode (DSI_HandleTypeDef * hdsi, uint32_t Lane, FunctionalState State)

Function description

Force the Clock/Data Lane in TX Stop Mode.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.
- **Lane:** select between clock or data lanes. This parameter can be any value of
  – DSI_Lane_Group
- **State:** ENABLE or DISABLE

Return values

- **HAL:** status

HAL_DSI_GetError

Function name

uint32_t HAL_DSI_GetError (DSI_HandleTypeDef * hdsi)

Function description

Return the DSI error code.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **DSI:** Error Code

HAL_DSI_GetState

Function name

HAL_DSI_StateTypeDef HAL_DSI_GetState (DSI_HandleTypeDef * hdsi)

Function description

Return the DSI state.

Parameters

- **hdsi:** pointer to a DSI_HandleTypeDef structure that contains the configuration information for the DSI.

Return values

- **HAL:** state

23.3 DSI Firmware driver defines

The following section lists the various define and macros of the module.

23.3.1 DSI

DSI

DSI Acknowledge Request

DSI_ACKNOWLEDGE_DISABLE

DSI_ACKNOWLEDGE_ENABLE

DSI Automatic Refresh

DSI_AR_DISABLE

DSI_AR_ENABLE

DSI Automatic Clk Lane Control

DSI_AUTO_CLK_LANE_CTRL_DISABLE

DSI_AUTO_CLK_LANE_CTRL_ENABLE

DSI Color Coding

DSI_RGB565

The values 0x00000001 and 0x00000002 can also be used for the RGB565 color mode configuration

DSI_RGB666

The value 0x00000004 can also be used for the RGB666 color mode configuration

DSI_RGB888

DSI Color Mode

DSI_COLOR_MODE_FULL

DSI_COLOR_MODE_EIGHT

DSI CustomLane

DSI_SWAP_LANE_PINS

DSI DATA ENABLE Polarity

DSI_DATA_ENABLE_ACTIVE_HIGH

DSI_DATA_ENABLE_ACTIVE_LOW

DSI DCS Command

DSI_ENTER_IDLE_MODE

DSI_ENTER_INVERT_MODE

DSI_ENTER_NORMAL_MODE

DSI_ENTER_PARTIAL_MODE

DSI_ENTER_SLEEP_MODE

DSI_EXIT_IDLE_MODE

DSI_EXIT_INVERT_MODE

DSI_EXIT_SLEEP_MODE

DSI_GET_3D_CONTROL

DSI_GET_ADDRESS_MODE

DSI_GET_BLUE_CHANNEL

DSI_GET_DIAGNOSTIC_RESULT

DSI_GET_DISPLAY_MODE

DSI_GET_GREEN_CHANNEL

DSI_GET_PIXEL_FORMAT

DSI_GET_POWER_MODE

DSI_GET_RED_CHANNEL

DSI_GET_SCANLINE

DSI_GET_SIGNAL_MODE

DSI_NOP

DSI_READ_DDB_CONTINUE

DSI_READ_DDB_START

DSI_READ_MEMORY_CONTINUE

DSI_READ_MEMORY_START

DSI_SET_3D_CONTROL

DSI_SET_ADDRESS_MODE

DSI_SET_COLUMN_ADDRESS

DSI_SET_DISPLAY_OFF

DSI_SET_DISPLAY_ON

DSI_SET_GAMMA_CURVE

DSI_SET_PAGE_ADDRESS

DSI_SET_PARTIAL_COLUMNS

DSI_SET_PARTIAL_ROWS

DSI_SET_PIXEL_FORMAT

DSI_SET_SCROLL_AREA

DSI_SET_SCROLL_START

DSI_SET_TEAR_OFF

DSI_SET_TEAR_ON

DSI_SET_TEAR_SCANLINE

DSI_SET_VSYNC_TIMING

DSI_SOFT_RESET

DSI_WRITE_LUT

DSI_WRITE_MEMORY_CONTINUE

DSI_WRITE_MEMORY_START

DSI D-PHY Slew rate

DSI_DPHY_SLEW_HS_TX_SPEED

80-750Mbits/s

DSI DPHY VCO Frequency Range

DSI_DPHY_VCO_FRANGE_500MHZ_800MHZ

500-800 MHz

DSI_DPHY_VCO_FRANGE_800MHZ_1GHZ

800-1000 MHz

DSI Error Data Type

HAL_DSI_ERROR_NONE

HAL_DSI_ERROR_ACK

Acknowledge errors

HAL_DSI_ERROR_PHY

PHY related errors

HAL_DSI_ERROR_TX

Transmission error

HAL_DSI_ERROR_RX

Reception error

HAL_DSI_ERROR_ECC

ECC errors

HAL_DSI_ERROR_CRC

CRC error

HAL_DSI_ERROR_PSE

Packet Size error

HAL_DSI_ERROR_EOT

End Of Transmission error

HAL_DSI_ERROR_OVF

FIFO overflow error

HAL_DSI_ERROR_GEN

Generic FIFO related errors

HAL_DSI_ERROR_PBU

Payload Buffer Underflow error

HAL_DSI_ERROR_INVALID_CALLBACK

DSI Invalid Callback error

DSI Exported Macros

__HAL_DSI_RESET_HANDLE_STATE

Description:

- Reset DSI handle state.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None

__HAL_DSI_ENABLE

Description:

- Enables the DSI host.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_DISABLE

Description:

- Disables the DSI host.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_WRAPPER_ENABLE

Description:

- Enables the DSI wrapper.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_WRAPPER_DISABLE

Description:

- Disable the DSI wrapper.

Parameters:

- __HANDLE__: DSI handle

Return value:

- None.

__HAL_DSI_PLL_ENABLE

Description:
- Enables the DSI PLL.

Parameters:
- __HANDLE__: DSI handle

Return value:
- None.

__HAL_DSI_PLL_DISABLE

Description:
- Disables the DSI PLL.

Parameters:
- __HANDLE__: DSI handle

Return value:
- None.

__HAL_DSI_GET_FLAG

Description:
- Get the DSI pending flags.

Parameters:
- __HANDLE__: DSI handle.
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  - DSI_FLAG_TE : Tearing Effect Interrupt Flag
  - DSI_FLAG_ER : End of Refresh Interrupt Flag
  - DSI_FLAG_BUSY : Busy Flag
  - DSI_FLAG_PLLLS: PLL Lock Status
  - DSI_FLAG_PLLL : PLL Lock Interrupt Flag
  - DSI_FLAG_PLLU : PLL Unlock Interrupt Flag
  - DSI_FLAG_RRS : Regulator Ready Flag
  - DSI_FLAG_RR : Regulator Ready Interrupt Flag

Return value:
- The: state of FLAG (SET or RESET).

__HAL_DSI_CLEAR_FLAG

Description:
- Clears the DSI pending flags.

Parameters:
- __HANDLE__: DSI handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - DSI_FLAG_TE : Tearing Effect Interrupt Flag
  - DSI_FLAG_ER : End of Refresh Interrupt Flag
  - DSI_FLAG_PLLL : PLL Lock Interrupt Flag
  - DSI_FLAG_PLLU : PLL Unlock Interrupt Flag
  - DSI_FLAG_RR : Regulator Ready Interrupt Flag

Return value:
- None

__HAL_DSI_ENABLE_IT

Description:

- Enables the specified DSI interrupts.

Parameters:

- __HANDLE__: DSI handle.
- __INTERRUPT__: specifies the DSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  – DSI_IT_TE : Tearing Effect Interrupt
  – DSI_IT_ER : End of Refresh Interrupt
  – DSI_IT_PLLL: PLL Lock Interrupt
  – DSI_IT_PLLU: PLL Unlock Interrupt
  – DSI_IT_RR : Regulator Ready Interrupt

Return value:

- None

__HAL_DSI_DISABLE_IT

Description:

- Disables the specified DSI interrupts.

Parameters:

- __HANDLE__: DSI handle
- __INTERRUPT__: specifies the DSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  – DSI_IT_TE : Tearing Effect Interrupt
  – DSI_IT_ER : End of Refresh Interrupt
  – DSI_IT_PLLL: PLL Lock Interrupt
  – DSI_IT_PLLU: PLL Unlock Interrupt
  – DSI_IT_RR : Regulator Ready Interrupt

Return value:

- None

__HAL_DSI_GET_IT_SOURCE

Description:

- Checks whether the specified DSI interrupt source is enabled or not.

Parameters:

- __HANDLE__: DSI handle
- __INTERRUPT__: specifies the DSI interrupt source to check. This parameter can be one of the following values:
  – DSI_IT_TE : Tearing Effect Interrupt
  – DSI_IT_ER : End of Refresh Interrupt
  – DSI_IT_PLLL: PLL Lock Interrupt
  – DSI_IT_PLLU: PLL Unlock Interrupt
  – DSI_IT_RR : Regulator Ready Interrupt

Return value:

- The: state of INTERRUPT (SET or RESET).

DSI FBTA Acknowledge

DSI_FBTAA_DISABLE

DSI_FBTAA_ENABLE

DSI Flags

DSI_FLAG_TE

DSI_FLAG_ER

DSI_FLAG_BUSY

DSI_FLAG_PLLLS

DSI_FLAG_PLLL

DSI_FLAG_PLLU

DSI Flow Control

DSI_FLOW_CONTROL_EOTP_TX_LP

DSI_FLOW_CONTROL_CRC_RX

DSI_FLOW_CONTROL_ECC_RX

DSI_FLOW_CONTROL_BTA

DSI_FLOW_CONTROL_EOTP_RX

DSI_FLOW_CONTROL_EOTP_TX

DSI_FLOW_CONTROL_ALL

DSI HSYNC Polarity

DSI_HSYNC_ACTIVE_HIGH

DSI_HSYNC_ACTIVE_LOW

DSI HS Prepare Offset

DSI_HS_PREPARE_OFFSET0

100-120 & 120-160 & 240-320 MHz

DSI_HS_PREPARE_OFFSET1

80-100 & 160-200 & 200-240 & 320-390 MHz

DSI_HS_PREPARE_OFFSET2

390-450 & 450-510 MHz

DSI HS Presp Mode

DSI_HS_PM_DISABLE

DSI_HS_PM_ENABLE

DSI Interrupts

DSI_IT_TE

DSI_IT_ER

DSI_IT_PLLL

DSI_IT_PLLU

DSI Lane Group

DSI_CLOCK_LANE

DSI_DATA_LANES

DSI Lane Select

DSI_CLK_LANE

DSI_DATA_LANE0

DSI_DATA_LANE1

DSI LONG WRITE PKT Data Type

DSI_DCS_LONG_PKT_WRITE

DCS long write

DSI_GEN_LONG_PKT_WRITE

Generic long write

DSI Loosely Packed

DSI_LOOSELY_PACKED_ENABLE

DSI_LOOSELY_PACKED_DISABLE

DSI LP Command

DSI_LP_COMMAND_DISABLE

DSI_LP_COMMAND_ENABLE

DSI LP HBP

DSI_LP_HBP_DISABLE

DSI_LP_HBP_ENABLE

DSI LP HFP

DSI_LP_HFP_DISABLE

DSI_LP_HFP_ENABLE

DSI LP LPDcs Long Write

DSI_LP_DLW_DISABLE

DSI_LP_DLW_ENABLE

DSI LP LPDcs Short Read NoP

DSI_LP_DSR0P_DISABLE

DSI_LP_DSR0P_ENABLE

DSI LP LPDcs Short Write NoP

DSI_LP_DSW0P_DISABLE

DSI_LP_DSW0P_ENABLE

DSI LP LPDcs Short Write OneP

DSI_LP_DSW1P_DISABLE

DSI_LP_DSW1P_ENABLE

DSI LP LPGen LongWrite

DSI_LP_GLW_DISABLE

DSI_LP_GLW_ENABLE

DSI LP LPGen Short Read NoP

DSI_LP_GSR0P_DISABLE

DSI_LP_GSR0P_ENABLE

DSI LP LPGen Short Read OneP

DSI_LP_GSR1P_DISABLE

DSI_LP_GSR1P_ENABLE

DSI LP LPGen Short Read TwoP

DSI_LP_GSR2P_DISABLE

DSI_LP_GSR2P_ENABLE

DSI LP LPGen Short Write NoP

DSI_LP_GSW0P_DISABLE

DSI_LP_GSW0P_ENABLE

DSI LP LPGen Short Write OneP

DSI_LP_GSW1P_DISABLE

DSI_LP_GSW1P_ENABLE

DSI LP LPGen Short Write TwoP

DSI_LP_GSW2P_DISABLE

DSI_LP_GSW2P_ENABLE

DSI LP LPMax Read Packet

DSI_LP_MRDP_DISABLE

DSI_LP_MRDP_ENABLE

DSI LP VACT

DSI_LP_VACT_DISABLE

DSI_LP_VACT_ENABLE

DSI LP VBP

DSI_LP_VBP_DISABLE

DSI_LP_VBP_ENABLE

DSI LP VFP

DSI_LP_VFP_DISABLE

DSI_LP_VFP_ENABLE

DSI LP VSYNC

DSI_LP_VSYNC_DISABLE

DSI_LP_VSYNC_ENABLE

DSI Number Of Lanes

DSI_ONE_DATA_LANE

DSI_TWO_DATA_LANES

DSI DPHY Frequency Range

DSI_DPHY_FRANGE_80MHZ_100MHZ

80-100 MHz

DSI_DPHY_FRANGE_100MHZ_120MHZ

100-120 MHz

DSI_DPHY_FRANGE_120MHZ_160MHZ

120-160 MHz

DSI_DPHY_FRANGE_160MHZ_200MHZ

160-200 MHz

DSI_DPHY_FRANGE_200MHZ_240MHZ
200-240 MHz

DSI_DPHY_FRANGE_240MHZ_320MHZ
240-320 MHz

DSI_DPHY_FRANGE_320MHZ_390MHZ
320-390 MHz

DSI_DPHY_FRANGE_390MHZ_450MHZ
390-450 MHz

DSI_DPHY_FRANGE_450MHZ_510MHZ
450-510 MHz

DSI PHY Low Power Offset

PHY_LP_OFFSSET_0_CLKP

PHY_LP_OFFSSET_1_CLKP

PHY_LP_OFFSSET_2_CLKP

PHY_LP_OFFSSET_3_CLKP

PHY_LP_OFFSSET_4_CLKP

PHY_LP_OFFSSET_5_CLKP

PHY_LP_OFFSSET_6_CLKP

PHY_LP_OFFSSET_7_CLKP

PHY_LP_OFFSSET_MINUS_1_CLKP

PHY_LP_OFFSSET_MINUS_2_CLKP

PHY_LP_OFFSSET_MINUS_3_CLKP

PHY_LP_OFFSSET_MINUS_4_CLKP

PHY_LP_OFFSSET_MINUS_5_CLKP

PHY_LP_OFFSSET_MINUS_6_CLKP

PHY_LP_OFFSSET_MINUS_7_CLKP

PHY_LP_OFFSSET_MINUS_8_CLKP

DSI PLL Charge pump

DSI_PLL_CHARGE_PUMP_2000HZ_4400HZ
2.0 to 4.4 MHz

DSI_PLL_CHARGE_PUMP_4400HZ_14100HZ
4.4 to 14.1 MHz

DSI_PLL_CHARGE_PUMP_14100HZ_30900HZ

14.1 to 30.9 MHz

DSI_PLL_CHARGE_PUMP_30900HZ_45700HZ

30.9 to 45.7 MHz

DSI_PLL_CHARGE_PUMP_45700HZ_50000HZ

45.7 to 50 MHz

DSI PLL IDF

DSI_PLL_IN_DIV1

DSI_PLL_IN_DIV2

DSI_PLL_IN_DIV3

DSI_PLL_IN_DIV4

DSI_PLL_IN_DIV5

DSI_PLL_IN_DIV6

DSI_PLL_IN_DIV7

DSI PLL ODF

DSI_PLL_OUT_DIV1

DSI_PLL_OUT_DIV2

DSI_PLL_OUT_DIV3

DSI_PLL_OUT_DIV4

DSI_PLL_OUT_DIV5

DSI_PLL_OUT_DIV6

DSI_PLL_OUT_DIV7

DSI_PLL_OUT_DIV8

DSI_PLL_OUT_DIV9

DSI PLL Tuning

DSI_PLL_LOOP_FILTER_2000HZ_4400HZ

2.0 to 4.4 MHz

DSI_PLL_LOOP_FILTER_4400HZ_30900HZ

4.4 to 30.9 MHz

DSI_PLL_LOOP_FILTER_30900HZ_50000HZ

30.9 to 50 MHz

DSI SHORT READ PKT Data Type

DSI_DCS_SHORT_PKT_READ

DCS short read

DSI_GEN_SHORT_PKT_READ_P0

Generic short read, no parameters

DSI_GEN_SHORT_PKT_READ_P1

Generic short read, one parameter

DSI_GEN_SHORT_PKT_READ_P2

Generic short read, two parameters

DSI SHORT WRITE PKT Data Type

DSI_DCS_SHORT_PKT_WRITE_P0

DCS short write, no parameters

DSI_DCS_SHORT_PKT_WRITE_P1

DCS short write, one parameter

DSI_GEN_SHORT_PKT_WRITE_P0

Generic short write, no parameters

DSI_GEN_SHORT_PKT_WRITE_P1

Generic short write, one parameter

DSI_GEN_SHORT_PKT_WRITE_P2

Generic short write, two parameters

DSI ShutDown

DSI_DISPLAY_ON

DSI_DISPLAY_OFF

DSI Tearing Effect Polarity

DSI_TE_RISING_EDGE

DSI_TE_FALLING_EDGE

DSI Tearing Effect Source

DSI_TE_DSILINK

DSI_TE_EXTERNAL

DSI TE Acknowledge Request

DSI_TE_ACKNOWLEDGE_DISABLE

DSI_TE_ACKNOWLEDGE_ENABLE

DSI Video Mode Type

DSI_VID_MODE_NB_PULSES

DSI_VID_MODE_NB_EVENTS

DSI_VID_MODE_BURST

DSI VSYNC Active Polarity

DSI_VSYNC_ACTIVE_HIGH

DSI_VSYNC_ACTIVE_LOW

DSI Vsync Polarity

DSI_VSYNC_FALLING

DSI_VSYNC_RISING

24 HAL EXTI Generic Driver

24.1 EXTI Firmware driver registers structures

24.1.1 EXTI_HandleTypeDef

EXTI_HandleTypeDef is defined in the stm32u5xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***void(* RisingCallback***
- ***void(* FallingCallback***

Field Documentation

- ***uint32_t EXTI_HandleTypeDef::Line***
  Exti line number
- ***void(* EXTI_HandleTypeDef::RisingCallback)(void)***
  Exti rising callback
- ***void(* EXTI_HandleTypeDef::FallingCallback)(void)***
  Exti falling callback

24.1.2 EXTI_ConfigTypeDef

EXTI_ConfigTypeDef is defined in the stm32u5xx_hal_exti.h

Data Fields

- ***uint32_t Line***
- ***uint32_t Mode***
- ***uint32_t Trigger***
- ***uint32_t GPIOSel***

Field Documentation

- ***uint32_t EXTI_ConfigTypeDef::Line***
  The Exti line to be configured. This parameter can be a value of ***EXTI_Line***
- ***uint32_t EXTI_ConfigTypeDef::Mode***
  The Exit Mode to be configured for a core. This parameter can be a combination of ***EXTI_Mode***
- ***uint32_t EXTI_ConfigTypeDef::Trigger***
  The Exti Trigger to be configured. This parameter can be a value of ***EXTI_Trigger***
- ***uint32_t EXTI_ConfigTypeDef::GPIOSel***
  The Exti GPIO multiplexer selection to be configured. This parameter is only possible for line 0 to 15. It can be a value of ***EXTI_GPIOSel***

24.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

24.2.1 EXTI Peripheral features

- Each Exti line can be configured within this driver.
- Exti line can be configured in 3 different modes
  - Interrupt
  - Event
  - Both of them
- Configurable Exti lines can be configured with 3 different triggers
  - Rising
  - Falling
  - Both of them

- When set in interrupt mode, configurable Exti lines have two diffenrents interrupt pending registers which allow to distinguish which transition occurs:
  - Rising edge pending interrupt
  - Falling
- Exti lines 0 to 15 are linked to gpio pin number 0 to 15. Gpio port can be selected through multiplexer.

24.2.2 How to use this driver

1. Configure the EXTI line using HAL_EXTI_SetConfigLine().
   - Choose the interrupt line number by setting "Line" member from EXTI_ConfigTypeDef structure.
   - Configure the interrupt and/or event mode using "Mode" member from EXTI_ConfigTypeDef structure.
   - For configurable lines, configure rising and/or falling trigger "Trigger" member from EXTI_ConfigTypeDef structure.
   - For Exti lines linked to gpio, choose gpio port using "GPIOSel" member from GPIO_InitTypeDef structure.
2. Get current Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
   - Provide pointer on EXTI_ConfigTypeDef structure as second parameter.
3. Clear Exti configuration of a dedicated line using HAL_EXTI_GetConfigLine().
   - Provide exiting handle as parameter.
4. Register callback to treat Exti interrupts using HAL_EXTI_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from EXTI_CallbackIDTypeDef.
   - Provide callback function pointer.
5. Get interrupt pending bit using HAL_EXTI_GetPending().
6. Clear interrupt pending bit using HAL_EXTI_GetPending().
7. Generate software interrupt using HAL_EXTI_GenerateSWI().

24.2.3 EXTI attributes functions

This section contains the following APIs:

- *HAL_EXTI_ConfigLineAttributes()*
- *HAL_EXTI_GetConfigLineAttributes()*
- *HAL_EXTI_LockAttributes()*
- *HAL_EXTI_GetLockAttributes()*

24.2.4 Configuration functions

This section contains the following APIs:

- *HAL_EXTI_SetConfigLine()*
- *HAL_EXTI_GetConfigLine()*
- *HAL_EXTI_ClearConfigLine()*
- *HAL_EXTI_RegisterCallback()*
- *HAL_EXTI_GetHandle()*

24.2.5 Detailed description of functions

HAL_EXTI_SetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_SetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Set configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on EXTI configuration to be set.

Return values

- **HAL:** Status.

HAL_EXTI_GetConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLine (EXTI_HandleTypeDef * hexti, EXTI_ConfigTypeDef * pExtiConfig)

Function description

Get configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.
- **pExtiConfig:** Pointer on structure to store Exti configuration.

Return values

- **HAL:** Status.

HAL_EXTI_ClearConfigLine

Function name

HAL_StatusTypeDef HAL_EXTI_ClearConfigLine (const EXTI_HandleTypeDef * hexti)

Function description

Clear whole configuration of a dedicated Exti line.

Parameters

- **hexti:** Exti handle.

Return values

- **HAL:** Status.

HAL_EXTI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_EXTI_RegisterCallback (EXTI_HandleTypeDef * hexti, EXTI_CallbackIDTypeDef CallbackID, void(*)(void) pPendingCbfn)

Function description

Register callback for a dedicaated Exti line.

Parameters

- **hexti:** Exti handle.
- **CallbackID:** User callback identifier. This parameter can be one of
  - EXTI_CallbackIDTypeDef values.
- **pPendingCbfn:** function pointer to be stored as callback.

Return values

- **HAL:** Status.

HAL_EXTI_GetHandle

Function name

HAL_StatusTypeDef HAL_EXTI_GetHandle (EXTI_HandleTypeDef * hexti, uint32_t ExtiLine)

Function description

Store line number as handle private field.

Parameters

- **hexti:** Exti handle.
- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.

Return values

- **HAL:** Status.

HAL_EXTI_IRQHandler

Function name

void HAL_EXTI_IRQHandler (const EXTI_HandleTypeDef * hexti)

Function description

Handle EXTI interrupt request.

Parameters

- **hexti:** Exti handle.

Return values

- **none.:**

HAL_EXTI_GetPending

Function name

uint32_t HAL_EXTI_GetPending (const EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Get interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be checked. This parameter can be one of the following values:
  - EXTI_TRIGGER_RISING
  - EXTI_TRIGGER_FALLING

Return values

- **1:** if interrupt is pending else 0.

HAL_EXTI_ClearPending

Function name

void HAL_EXTI_ClearPending (const EXTI_HandleTypeDef * hexti, uint32_t Edge)

Function description

Clear interrupt pending bit of a dedicated line.

Parameters

- **hexti:** Exti handle.
- **Edge:** Specify which pending edge as to be clear. This parameter can be one of the following values:
  – EXTI_TRIGGER_RISING
  – EXTI_TRIGGER_FALLING

Return values

- **None.:**

HAL_EXTI_GenerateSWI

Function name

void HAL_EXTI_GenerateSWI (const EXTI_HandleTypeDef * hexti)

Function description

Generate a software interrupt for a dedicated line.

Parameters

- **hexti:** Exti handle.

Return values

- **None.:**

HAL_EXTI_ConfigLineAttributes

Function name

void HAL_EXTI_ConfigLineAttributes (uint32_t ExtiLine, uint32_t LineAttributes)

Function description

Configure the EXTI line attribute(s).

Parameters

- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.
- **LineAttributes:** can be one or a combination of the following values:
  – EXTI_LINE_PRIV Privileged-only access
  – EXTI_LINE_NPRIV Privileged/Non-privileged access
  – EXTI_LINE_SEC Secure-only access
  – EXTI_LINE_NSEC Secure/Non-secure access

Return values

- **None:**

Notes

- Available attributes are to secure EXTI line and set EXT line as privileged. Default state is not secure and unprivileged access allowed.
- Secure and non-secure attributes can only be set from the secure state when the system implements the security (TZEN=1).
- Security and privilege attributes can be set independently.

HAL_EXTI_GetConfigLineAttributes

Function name

HAL_StatusTypeDef HAL_EXTI_GetConfigLineAttributes (uint32_t ExtiLine, uint32_t * pLineAttributes)

Function description

Get the EXTI line attribute(s).

Parameters

- **ExtiLine:** Exti line number. This parameter can be from 0 to EXTI_LINE_NB.
- **pLineAttributes:** pointer to return line attributes.

Return values

- **HAL:** Status.

Notes

- Secure and non-secure attributes are only available from secure state when the system implements the security (TZEN=1)

HAL_EXTI_LockAttributes

Function name

void HAL_EXTI_LockAttributes (void)

Function description

Lock the secure and privilege configuration registers.

Return values

- **None:**

HAL_EXTI_GetLockAttributes

Function name

uint32_t HAL_EXTI_GetLockAttributes (void)

Function description

Return the secure and privilege configuration registers LOCK status.

Return values

- **1:** if the secure and privilege configuration registers have been locked else 0.

24.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

24.3.1 EXTI

EXTI

EXTI GPIOSel

EXTI_GPIOA

EXTI_GPIOB

EXTI_GPIOC

EXTI_GPIOD

EXTI_GPIOE

EXTI_GPIOF

EXTI_GPIOG

EXTI_GPIOH

EXTI_GPIOI

EXTI_GPIOJ

EXTI Line

EXTI_LINE_0

EXTI_LINE_1

EXTI_LINE_2

EXTI_LINE_3

EXTI_LINE_4

EXTI_LINE_5

EXTI_LINE_6

EXTI_LINE_7

EXTI_LINE_8

EXTI_LINE_9

EXTI_LINE_10

EXTI_LINE_11

EXTI_LINE_12

EXTI_LINE_13

EXTI_LINE_14

EXTI_LINE_15

EXTI_LINE_16

EXTI_LINE_17

EXTI_LINE_18

EXTI_LINE_19

EXTI_LINE_20

EXTI_LINE_21

EXTI_LINE_22

EXTI_LINE_23

EXTI_LINE_24

EXTI_LINE_25

EXTI line attributes

EXTI_LINE_SEC

< Secure line attribute Non-secure line attribute

EXTI_LINE_NSEC

EXTI_LINE_PRIV

< Privileged line attribute Non-privileged line attribute

EXTI_LINE_NPRIV

EXTI Mode

EXTI_MODE_NONE

EXTI_MODE_INTERRUPT

EXTI_MODE_EVENT

EXTI Trigger

EXTI_TRIGGER_NONE

EXTI_TRIGGER_RISING

EXTI_TRIGGER_FALLING

EXTI_TRIGGER_RISING_FALLING

25 HAL FDCAN Generic Driver

25.1 FDCAN Firmware driver registers structures

25.1.1 FDCAN_InitTypeDef

FDCAN_InitTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t ClockDivider***
- ***uint32_t FrameFormat***
- ***uint32_t Mode***
- ***FunctionalState AutoRetransmission***
- ***FunctionalState TransmitPause***
- ***FunctionalState ProtocolException***
- ***uint32_t NominalPrescaler***
- ***uint32_t NominalSyncJumpWidth***
- ***uint32_t NominalTimeSeg1***
- ***uint32_t NominalTimeSeg2***
- ***uint32_t DataPrescaler***
- ***uint32_t DataSyncJumpWidth***
- ***uint32_t DataTimeSeg1***
- ***uint32_t DataTimeSeg2***
- ***uint32_t StdFiltersNbr***
- ***uint32_t ExtFiltersNbr***
- ***uint32_t TxFifoQueueMode***

Field Documentation

- ***uint32_t FDCAN_InitTypeDef::ClockDivider***
  Specifies the FDCAN kernel clock divider. The clock is common to all FDCAN instances. This parameter is applied only at initialisation of first FDCAN instance. This parameter can be a value of ***FDCAN_clock_divider***.
- ***uint32_t FDCAN_InitTypeDef::FrameFormat***
  Specifies the FDCAN frame format. This parameter can be a value of ***FDCAN_frame_format***
- ***uint32_t FDCAN_InitTypeDef::Mode***
  Specifies the FDCAN mode. This parameter can be a value of ***FDCAN_operating_mode***
- ***FunctionalState FDCAN_InitTypeDef::AutoRetransmission***
  Enable or disable the automatic retransmission mode. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::TransmitPause***
  Enable or disable the Transmit Pause feature. This parameter can be set to ENABLE or DISABLE
- ***FunctionalState FDCAN_InitTypeDef::ProtocolException***
  Enable or disable the Protocol Exception Handling. This parameter can be set to ENABLE or DISABLE
- ***uint32_t FDCAN_InitTypeDef::NominalPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the nominal bit time quanta. This parameter must be a number between 1 and 512
- ***uint32_t FDCAN_InitTypeDef::NominalSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a bit to perform resynchronization. This parameter must be a number between 1 and 128
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg1***
  Specifies the number of time quanta in Bit Segment 1. This parameter must be a number between 2 and 256
- ***uint32_t FDCAN_InitTypeDef::NominalTimeSeg2***
  Specifies the number of time quanta in Bit Segment 2. This parameter must be a number between 2 and 128

- ***uint32_t FDCAN_InitTypeDef::DataPrescaler***
  Specifies the value by which the oscillator frequency is divided for generating the data bit time quanta. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataSyncJumpWidth***
  Specifies the maximum number of time quanta the FDCAN hardware is allowed to lengthen or shorten a data bit to perform resynchronization. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg1***
  Specifies the number of time quanta in Data Bit Segment 1. This parameter must be a number between 1 and 32
- ***uint32_t FDCAN_InitTypeDef::DataTimeSeg2***
  Specifies the number of time quanta in Data Bit Segment 2. This parameter must be a number between 1 and 16
- ***uint32_t FDCAN_InitTypeDef::StdFiltersNbr***
  Specifies the number of standard Message ID filters. This parameter must be a number between 0 and 28
- ***uint32_t FDCAN_InitTypeDef::ExtFiltersNbr***
  Specifies the number of extended Message ID filters. This parameter must be a number between 0 and 8
- ***uint32_t FDCAN_InitTypeDef::TxFifoQueueMode***
  Tx FIFO/Queue Mode selection. This parameter can be a value of ***FDCAN_txFifoQueue_Mode***

25.1.2 FDCAN_FilterTypeDef

FDCAN_FilterTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t IdType***
- ***uint32_t FilterIndex***
- ***uint32_t FilterType***
- ***uint32_t FilterConfig***
- ***uint32_t FilterID1***
- ***uint32_t FilterID2***

Field Documentation

- ***uint32_t FDCAN_FilterTypeDef::IdType***
  Specifies the identifier type. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_FilterTypeDef::FilterIndex***
  Specifies the filter which will be initialized. This parameter must be a number between:
  – 0 and (SRAMCAN_FLS_NBR-1), if IdType is FDCAN_STANDARD_ID
  – 0 and (SRAMCAN_FLE_NBR-1), if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterType***
  Specifies the filter type. This parameter can be a value of ***FDCAN_filter_type***. The value FDCAN_FILTER_RANGE_NO_EIDM is permitted only when IdType is FDCAN_EXTENDED_ID.
- ***uint32_t FDCAN_FilterTypeDef::FilterConfig***
  Specifies the filter configuration. This parameter can be a value of ***FDCAN_filter_config***
- ***uint32_t FDCAN_FilterTypeDef::FilterID1***
  Specifies the filter identification 1. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_FilterTypeDef::FilterID2***
  Specifies the filter identification 2. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID

25.1.3 FDCAN_TxHeaderTypeDef

FDCAN_TxHeaderTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***

- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxEventFifoControl***
- ***uint32_t MessageMarker***

Field Documentation

- ***uint32_t FDCAN_TxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxHeaderTypeDef::IdType***
  Specifies the identifier type for the message that will be transmitted. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxFrameType***
  Specifies the frame type of the message that will be transmitted. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxHeaderTypeDef::DataLength***
  Specifies the length of the frame that will be transmitted. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Tx frame will be transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxHeaderTypeDef::FDFormat***
  Specifies whether the Tx frame will be transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxHeaderTypeDef::TxEventFifoControl***
  Specifies the event FIFO control. This parameter can be a value of ***Section 25.3.1: FDCAN_EFC***
- ***uint32_t FDCAN_TxHeaderTypeDef::MessageMarker***
  Specifies the message marker to be copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF

25.1.4 FDCAN_RxHeaderTypeDef

FDCAN_RxHeaderTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t RxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t RxTimestamp***
- ***uint32_t FilterIndex***
- ***uint32_t IsFilterMatchingFrame***

Field Documentation

- ***uint32_t FDCAN_RxHeaderTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_RxHeaderTypeDef::IdType***
  Specifies the identifier type of the received message. This parameter can be a value of ***FDCAN_id_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxFrameType***
  Specifies the the received message frame type. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_RxHeaderTypeDef::DataLength***
  Specifies the received frame length. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_RxHeaderTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_RxHeaderTypeDef::BitRateSwitch***
  Specifies whether the Rx frame is received with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_RxHeaderTypeDef::FDFormat***
  Specifies whether the Rx frame is received in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_RxHeaderTypeDef::RxTimestamp***
  Specifies the timestamp counter value captured on start of frame reception. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_RxHeaderTypeDef::FilterIndex***
  Specifies the index of matching Rx acceptance filter element. This parameter must be a number between:
  – 0 and (SRAMCAN_FLS_NBR-1), if IdType is FDCAN_STANDARD_ID
  – 0 and (SRAMCAN_FLE_NBR-1), if IdType is FDCAN_EXTENDED_ID When the frame is a Non-Filter matching frame, this parameter is unused.
- ***uint32_t FDCAN_RxHeaderTypeDef::IsFilterMatchingFrame***
  Specifies whether the accepted frame did not match any Rx filter. Acceptance of non-matching frames may be enabled via **HAL_FDCAN_ConfigGlobalFilter()**. This parameter takes 0 if the frame matched an Rx filter or 1 if it did not match any Rx filter

25.1.5 FDCAN_TxEventFifoTypeDef

FDCAN_TxEventFifoTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t Identifier***
- ***uint32_t IdType***
- ***uint32_t TxFrameType***
- ***uint32_t DataLength***
- ***uint32_t ErrorStateIndicator***
- ***uint32_t BitRateSwitch***
- ***uint32_t FDFormat***
- ***uint32_t TxTimestamp***
- ***uint32_t MessageMarker***
- ***uint32_t EventType***

Field Documentation

- ***uint32_t FDCAN_TxEventFifoTypeDef::Identifier***
  Specifies the identifier. This parameter must be a number between:
  – 0 and 0x7FF, if IdType is FDCAN_STANDARD_ID
  – 0 and 0x1FFFFFFF, if IdType is FDCAN_EXTENDED_ID
- ***uint32_t FDCAN_TxEventFifoTypeDef::IdType***
  Specifies the identifier type for the transmitted message. This parameter can be a value of ***FDCAN_id_type***

- ***uint32_t FDCAN_TxEventFifoTypeDef::TxFrameType***
  Specifies the frame type of the transmitted message. This parameter can be a value of ***FDCAN_frame_type***
- ***uint32_t FDCAN_TxEventFifoTypeDef::DataLength***
  Specifies the length of the transmitted frame. This parameter can be a value of ***FDCAN_data_length_code***
- ***uint32_t FDCAN_TxEventFifoTypeDef::ErrorStateIndicator***
  Specifies the error state indicator. This parameter can be a value of ***FDCAN_error_state_indicator***
- ***uint32_t FDCAN_TxEventFifoTypeDef::BitRateSwitch***
  Specifies whether the Tx frame is transmitted with or without bit rate switching. This parameter can be a value of ***FDCAN_bit_rate_switching***
- ***uint32_t FDCAN_TxEventFifoTypeDef::FDFormat***
  Specifies whether the Tx frame is transmitted in classic or FD format. This parameter can be a value of ***FDCAN_format***
- ***uint32_t FDCAN_TxEventFifoTypeDef::TxTimestamp***
  Specifies the timestamp counter value captured on start of frame transmission. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::MessageMarker***
  Specifies the message marker copied into Tx Event FIFO element for identification of Tx message status. This parameter must be a number between 0 and 0xFF
- ***uint32_t FDCAN_TxEventFifoTypeDef::EventType***
  Specifies the event type. This parameter can be a value of ***FDCAN_event_type***

25.1.6 FDCAN_HpMsgStatusTypeDef

FDCAN_HpMsgStatusTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t FilterList***
- ***uint32_t FilterIndex***
- ***uint32_t MessageStorage***
- ***uint32_t MessageIndex***

Field Documentation

- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterList***
  Specifies the filter list of the matching filter element. This parameter can be:
  – 0 : Standard Filter List
  – 1 : Extended Filter List
- ***uint32_t FDCAN_HpMsgStatusTypeDef::FilterIndex***
  Specifies the index of matching filter element. This parameter can be a number between:
  – 0 and (SRAMCAN_FLS_NBR-1), if FilterList is 0 (Standard)
  – 0 and (SRAMCAN_FLE_NBR-1), if FilterList is 1 (Extended)
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageStorage***
  Specifies the HP Message Storage. This parameter can be a value of ***FDCAN_hp_msg_storage***
- ***uint32_t FDCAN_HpMsgStatusTypeDef::MessageIndex***
  Specifies the Index of Rx FIFO element to which the message was stored. This parameter is valid only when MessageStorage is: FDCAN_HP_STORAGE_RXFIFO0 or FDCAN_HP_STORAGE_RXFIFO1

25.1.7 FDCAN_ProtocolStatusTypeDef

FDCAN_ProtocolStatusTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t LastErrorCode***
- ***uint32_t DataLastErrorCode***
- ***uint32_t Activity***
- ***uint32_t ErrorPassive***
- ***uint32_t Warning***
- ***uint32_t BusOff***

- ***uint32_t RxESIflag***
- ***uint32_t RxBRSflag***
- ***uint32_t RxFDFflag***
- ***uint32_t ProtocolException***
- ***uint32_t TDCvalue***

Field Documentation

- ***uint32_t FDCAN_ProtocolStatusTypeDef::LastErrorCode***
  Specifies the type of the last error that occurred on the FDCAN bus. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::DataLastErrorCode***
  Specifies the type of the last error that occurred in the data phase of a CAN FD format frame with its BRS flag set. This parameter can be a value of ***FDCAN_protocol_error_code***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Activity***
  Specifies the FDCAN module communication state. This parameter can be a value of ***FDCAN_communication_state***
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ErrorPassive***
  Specifies the FDCAN module error status. This parameter can be:
  - 0 : The FDCAN is in Error_Active state
  - 1 : The FDCAN is in Error_Passive state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::Warning***
  Specifies the FDCAN module warning status. This parameter can be:
  - 0 : error counters (RxErrorCnt and TxErrorCnt) are below the Error_Warning limit of 96
  - 1 : at least one of error counters has reached the Error_Warning limit of 96
- ***uint32_t FDCAN_ProtocolStatusTypeDef::BusOff***
  Specifies the FDCAN module Bus_Off status. This parameter can be:
  - 0 : The FDCAN is not in Bus_Off state
  - 1 : The FDCAN is in Bus_Off state
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxESIflag***
  Specifies ESI flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its ESI flag set
  - 1 : Last received CAN FD message had its ESI flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxBRSflag***
  Specifies BRS flag of last received CAN FD message. This parameter can be:
  - 0 : Last received CAN FD message did not have its BRS flag set
  - 1 : Last received CAN FD message had its BRS flag set
- ***uint32_t FDCAN_ProtocolStatusTypeDef::RxFDFflag***
  Specifies if CAN FD message (FDF flag set) has been received since last protocol status. This parameter can be:
  - 0 : No CAN FD message received
  - 1 : CAN FD message received
- ***uint32_t FDCAN_ProtocolStatusTypeDef::ProtocolException***
  Specifies the FDCAN module Protocol Exception status. This parameter can be:
  - 0 : No protocol exception event occurred since last read access
  - 1 : Protocol exception event occurred
- ***uint32_t FDCAN_ProtocolStatusTypeDef::TDCvalue***
  Specifies the Transmitter Delay Compensation Value. This parameter can be a number between 0 and 127

25.1.8 FDCAN_ErrorCountersTypeDef

FDCAN_ErrorCountersTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t TxErrorCnt***
- ***uint32_t RxErrorCnt***
- ***uint32_t RxErrorPassive***

- ***uint32_t ErrorLogging***

Field Documentation

- ***uint32_t FDCAN_ErrorCountersTypeDef::TxErrorCnt***
  Specifies the Transmit Error Counter Value. This parameter can be a number between 0 and 255
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorCnt***
  Specifies the Receive Error Counter Value. This parameter can be a number between 0 and 127
- ***uint32_t FDCAN_ErrorCountersTypeDef::RxErrorPassive***
  Specifies the Receive Error Passive status. This parameter can be:
  - 0 : The Receive Error Counter (RxErrorCnt) is below the error passive level of 128
  - 1 : The Receive Error Counter (RxErrorCnt) has reached the error passive level of 128
- ***uint32_t FDCAN_ErrorCountersTypeDef::ErrorLogging***
  Specifies the Transmit/Receive error logging counter value. This parameter can be a number between 0 and 255. This counter is incremented each time when a FDCAN protocol error causes the TxErrorCnt or the RxErrorCnt to be incremented. The counter stops at 255; the next increment of TxErrorCnt or RxErrorCnt sets interrupt flag FDCAN_FLAG_ERROR_LOGGING_OVERFLOW

25.1.9 FDCAN_MsgRamAddressTypeDef

FDCAN_MsgRamAddressTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***uint32_t StandardFilterSA***
- ***uint32_t ExtendedFilterSA***
- ***uint32_t RxFIFO0SA***
- ***uint32_t RxFIFO1SA***
- ***uint32_t TxEventFIFOSA***
- ***uint32_t TxFIFOQSA***

Field Documentation

- ***uint32_t FDCAN_MsgRamAddressTypeDef::StandardFilterSA***
  Specifies the Standard Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::ExtendedFilterSA***
  Specifies the Extended Filter List Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO0SA***
  Specifies the Rx FIFO 0 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::RxFIFO1SA***
  Specifies the Rx FIFO 1 Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxEventFIFOSA***
  Specifies the Tx Event FIFO Start Address. This parameter must be a 32-bit word address
- ***uint32_t FDCAN_MsgRamAddressTypeDef::TxFIFOQSA***
  Specifies the Tx FIFO/Queue Start Address. This parameter must be a 32-bit word address

25.1.10 __FDCAN_HandleTypeDef

__FDCAN_HandleTypeDef is defined in the stm32u5xx_hal_fdcan.h

Data Fields

- ***FDCAN_GlobalTypeDef * Instance***
- ***FDCAN_InitTypeDef Init***
- ***FDCAN_MsgRamAddressTypeDef msgRam***
- ***uint32_t LatestTxFifoQRequest***
- ***__IO HAL_FDCAN_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* TxEventFifoCallback***
- ***void(* RxFifo0Callback***
- ***void(* RxFifo1Callback***

- *void(* TxFifoEmptyCallback*
- *void(* TxBufferCompleteCallback*
- *void(* TxBufferAbortCallback*
- *void(* HighPriorityMessageCallback*
- *void(* TimestampWraparoundCallback*
- *void(* TimeoutOccurredCallback*
- *void(* ErrorCallback*
- *void(* ErrorStatusCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***FDCAN_GlobalTypeDef* __FDCAN_HandleTypeDef::Instance***
  Register base address
- ***FDCAN_InitTypeDef __FDCAN_HandleTypeDef::Init***
  FDCAN required parameters
- ***FDCAN_MsgRamAddressTypeDef __FDCAN_HandleTypeDef::msgRam***
  FDCAN Message RAM blocks
- ***uint32_t __FDCAN_HandleTypeDef::LatestTxFifoQRequest***
  FDCAN Tx buffer index of latest Tx FIFO/Queue request
- ***__IO HAL_FDCAN_StateTypeDef __FDCAN_HandleTypeDef::State***
  FDCAN communication state
- ***HAL_LockTypeDef __FDCAN_HandleTypeDef::Lock***
  FDCAN locking object
- ***__IO uint32_t __FDCAN_HandleTypeDef::ErrorCode***
  FDCAN Error code
- ***void(* __FDCAN_HandleTypeDef::TxEventFifoCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t TxEventFifoITs)***
  FDCAN Tx Event Fifo callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo0Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo0ITs)***
  FDCAN Rx Fifo 0 callback
- ***void(* __FDCAN_HandleTypeDef::RxFifo1Callback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t RxFifo1ITs)***
  FDCAN Rx Fifo 1 callback
- ***void(* __FDCAN_HandleTypeDef::TxFifoEmptyCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Tx Fifo Empty callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferCompleteCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer complete callback
- ***void(* __FDCAN_HandleTypeDef::TxBufferAbortCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t BufferIndexes)***
  FDCAN Tx Buffer abort callback
- ***void(* __FDCAN_HandleTypeDef::HighPriorityMessageCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN High priority message callback
- ***void(* __FDCAN_HandleTypeDef::TimestampWraparoundCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timestamp wraparound callback
- ***void(* __FDCAN_HandleTypeDef::TimeoutOccurredCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Timeout occurred callback
- ***void(* __FDCAN_HandleTypeDef::ErrorCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Error callback

- ***void(* __FDCAN_HandleTypeDef::ErrorStatusCallback)(struct __FDCAN_HandleTypeDef *hfdcan, uint32_t ErrorStatusITs)***
  FDCAN Error status callback
- ***void(* __FDCAN_HandleTypeDef::MspInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp Init callback
- ***void(* __FDCAN_HandleTypeDef::MspDeInitCallback)(struct __FDCAN_HandleTypeDef *hfdcan)***
  FDCAN Msp DeInit callback

25.2 FDCAN Firmware driver API description

The following section lists the various functions of the FDCAN library.

25.2.1 How to use this driver

1. Initialize the FDCAN peripheral using HAL_FDCAN_Init function.
2. If needed , configure the reception filters and optional features using the following configuration functions:
   – HAL_FDCAN_ConfigFilter
   – HAL_FDCAN_ConfigGlobalFilter
   – HAL_FDCAN_ConfigExtendedIdMask
   – HAL_FDCAN_ConfigRxFifoOverwrite
   – HAL_FDCAN_ConfigRamWatchdog
   – HAL_FDCAN_ConfigTimestampCounter
   – HAL_FDCAN_EnableTimestampCounter
   – HAL_FDCAN_DisableTimestampCounter
   – HAL_FDCAN_ConfigTimeoutCounter
   – HAL_FDCAN_EnableTimeoutCounter
   – HAL_FDCAN_DisableTimeoutCounter
   – HAL_FDCAN_ConfigTxDelayCompensation
   – HAL_FDCAN_EnableTxDelayCompensation
   – HAL_FDCAN_DisableTxDelayCompensation
   – HAL_FDCAN_EnableISOMode
   – HAL_FDCAN_DisableISOMode
   – HAL_FDCAN_EnableEdgeFiltering
   – HAL_FDCAN_DisableEdgeFiltering
3. Start the FDCAN module using HAL_FDCAN_Start function. At this level the node is active on the bus: it can send and receive messages.
4. The following Tx control functions can only be called when the FDCAN module is started:
   – HAL_FDCAN_AddMessageToTxFifoQ
   – HAL_FDCAN_AbortTxRequest
5. After having submitted a Tx request in Tx Fifo or Queue, it is possible to get Tx buffer location used to place the Tx request thanks to HAL_FDCAN_GetLatestTxFifoQRequestBuffer API. It is then possible to abort later on the corresponding Tx Request using HAL_FDCAN_AbortTxRequest API.
6. When a message is received into the FDCAN message RAM, it can be retrieved using the HAL_FDCAN_GetRxMessage function.
7. Calling the HAL_FDCAN_Stop function stops the FDCAN module by entering it to initialization mode and re-enabling access to configuration registers through the configuration functions listed here above.
8. All other control functions can be called any time after initialization phase, no matter if the FDCAN module is started or stopped.

Polling mode operation

1. Reception and transmission states can be monitored via the following functions:
   – HAL_FDCAN_IsTxBufferMessagePending
   – HAL_FDCAN_GetRxFifoFillLevel
   – HAL_FDCAN_GetTxFifoFreeLevel

Interrupt mode operation

1. There are two interrupt lines: line 0 and 1. By default, all interrupts are assigned to line 0. Interrupt lines can be configured using HAL_FDCAN_ConfigInterruptLines function.
2. Notifications are activated using HAL_FDCAN_ActivateNotification function. Then, the process can be controlled through one of the available user callbacks: HAL_FDCAN_xxxCallback.

Callback registration

25.2.2 Initialization and de-initialization functions

This section provides functions allowing to:
- Initialize and configure the FDCAN.
- De-initialize the FDCAN.
- Enter FDCAN peripheral in power down mode.
- Exit power down mode.
- Register callbacks.
- Unregister callbacks.

This section contains the following APIs:
- *HAL_FDCAN_Init()*
- *HAL_FDCAN_DeInit()*
- *HAL_FDCAN_MspInit()*
- *HAL_FDCAN_MspDeInit()*
- *HAL_FDCAN_EnterPowerDownMode()*
- *HAL_FDCAN_ExitPowerDownMode()*
- *HAL_FDCAN_RegisterCallback()*
- *HAL_FDCAN_UnRegisterCallback()*
- *HAL_FDCAN_RegisterTxEventFifoCallback()*
- *HAL_FDCAN_UnRegisterTxEventFifoCallback()*
- *HAL_FDCAN_RegisterRxFifo0Callback()*
- *HAL_FDCAN_UnRegisterRxFifo0Callback()*
- *HAL_FDCAN_RegisterRxFifo1Callback()*
- *HAL_FDCAN_UnRegisterRxFifo1Callback()*
- *HAL_FDCAN_RegisterTxBufferCompleteCallback()*
- *HAL_FDCAN_UnRegisterTxBufferCompleteCallback()*
- *HAL_FDCAN_RegisterTxBufferAbortCallback()*
- *HAL_FDCAN_UnRegisterTxBufferAbortCallback()*
- *HAL_FDCAN_RegisterErrorStatusCallback()*
- *HAL_FDCAN_UnRegisterErrorStatusCallback()*
- *HAL_FDCAN_RegisterCallback()*

25.2.3 Configuration functions

This section provides functions allowing to:
- HAL_FDCAN_ConfigFilter : Configure the FDCAN reception filters
- HAL_FDCAN_ConfigGlobalFilter : Configure the FDCAN global filter
- HAL_FDCAN_ConfigExtendedIdMask : Configure the extended ID mask
- HAL_FDCAN_ConfigRxFifoOverwrite : Configure the Rx FIFO operation mode
- HAL_FDCAN_ConfigRamWatchdog : Configure the RAM watchdog
- HAL_FDCAN_ConfigTimestampCounter : Configure the timestamp counter
- HAL_FDCAN_EnableTimestampCounter : Enable the timestamp counter
- HAL_FDCAN_DisableTimestampCounter : Disable the timestamp counter

- HAL_FDCAN_GetTimestampCounter : Get the timestamp counter value
- HAL_FDCAN_ResetTimestampCounter : Reset the timestamp counter to zero
- HAL_FDCAN_ConfigTimeoutCounter : Configure the timeout counter
- HAL_FDCAN_EnableTimeoutCounter : Enable the timeout counter
- HAL_FDCAN_DisableTimeoutCounter : Disable the timeout counter
- HAL_FDCAN_GetTimeoutCounter : Get the timeout counter value
- HAL_FDCAN_ResetTimeoutCounter : Reset the timeout counter to its start value
- HAL_FDCAN_ConfigTxDelayCompensation : Configure the transmitter delay compensation
- HAL_FDCAN_EnableTxDelayCompensation : Enable the transmitter delay compensation
- HAL_FDCAN_DisableTxDelayCompensation : Disable the transmitter delay compensation
- HAL_FDCAN_EnableISOMode : Enable ISO 11898-1 protocol mode
- HAL_FDCAN_DisableISOMode : Disable ISO 11898-1 protocol mode
- HAL_FDCAN_EnableEdgeFiltering : Enable edge filtering during bus integration
- HAL_FDCAN_DisableEdgeFiltering : Disable edge filtering during bus integration

This section contains the following APIs:

- ***HAL_FDCAN_ConfigFilter()***
- ***HAL_FDCAN_ConfigGlobalFilter()***
- ***HAL_FDCAN_ConfigExtendedIdMask()***
- ***HAL_FDCAN_ConfigRxFifoOverwrite()***
- ***HAL_FDCAN_ConfigRamWatchdog()***
- ***HAL_FDCAN_ConfigTimestampCounter()***
- ***HAL_FDCAN_EnableTimestampCounter()***
- ***HAL_FDCAN_DisableTimestampCounter()***
- ***HAL_FDCAN_GetTimestampCounter()***
- ***HAL_FDCAN_ResetTimestampCounter()***
- ***HAL_FDCAN_ConfigTimeoutCounter()***
- ***HAL_FDCAN_EnableTimeoutCounter()***
- ***HAL_FDCAN_DisableTimeoutCounter()***
- ***HAL_FDCAN_GetTimeoutCounter()***
- ***HAL_FDCAN_ResetTimeoutCounter()***
- ***HAL_FDCAN_ConfigTxDelayCompensation()***
- ***HAL_FDCAN_EnableTxDelayCompensation()***
- ***HAL_FDCAN_DisableTxDelayCompensation()***
- ***HAL_FDCAN_EnableISOMode()***
- ***HAL_FDCAN_DisableISOMode()***
- ***HAL_FDCAN_EnableEdgeFiltering()***
- ***HAL_FDCAN_DisableEdgeFiltering()***

25.2.4 Control functions

This section provides functions allowing to:

- HAL_FDCAN_Start : Start the FDCAN module
- HAL_FDCAN_Stop : Stop the FDCAN module and enable access to configuration registers
- HAL_FDCAN_AddMessageToTxFifoQ : Add a message to the Tx FIFO/Queue and activate the corresponding transmission request
- HAL_FDCAN_GetLatestTxFifoQRequestBuffer : Get Tx buffer index of latest Tx FIFO/Queue request
- HAL_FDCAN_AbortTxRequest : Abort transmission request
- HAL_FDCAN_GetRxMessage : Get an FDCAN frame from the Rx FIFO zone into the message RAM
- HAL_FDCAN_GetTxEvent : Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM
- HAL_FDCAN_GetHighPriorityMessageStatus : Get high priority message status
- HAL_FDCAN_GetProtocolStatus : Get protocol status

- HAL_FDCAN_GetErrorCounters : Get error counter values
- HAL_FDCAN_IsTxBufferMessagePending : Check if a transmission request is pending on the selected Tx buffer
- HAL_FDCAN_GetRxFifoFillLevel : Return Rx FIFO fill level
- HAL_FDCAN_GetTxFifoFreeLevel : Return Tx FIFO free level
- HAL_FDCAN_IsRestrictedOperationMode : Check if the FDCAN peripheral entered Restricted Operation Mode
- HAL_FDCAN_ExitRestrictedOperationMode : Exit Restricted Operation Mode

This section contains the following APIs:
- ***HAL_FDCAN_Start()***
- ***HAL_FDCAN_Stop()***
- ***HAL_FDCAN_AddMessageToTxFifoQ()***
- ***HAL_FDCAN_GetLatestTxFifoQRequestBuffer()***
- ***HAL_FDCAN_AbortTxRequest()***
- ***HAL_FDCAN_GetRxMessage()***
- ***HAL_FDCAN_GetTxEvent()***
- ***HAL_FDCAN_GetHighPriorityMessageStatus()***
- ***HAL_FDCAN_GetProtocolStatus()***
- ***HAL_FDCAN_GetErrorCounters()***
- ***HAL_FDCAN_IsTxBufferMessagePending()***
- ***HAL_FDCAN_GetRxFifoFillLevel()***
- ***HAL_FDCAN_GetTxFifoFreeLevel()***
- ***HAL_FDCAN_IsRestrictedOperationMode()***
- ***HAL_FDCAN_ExitRestrictedOperationMode()***

25.2.5 Interrupts management

This section provides functions allowing to:
- HAL_FDCAN_ConfigInterruptLines : Assign interrupts to either Interrupt line 0 or 1
- HAL_FDCAN_ActivateNotification : Enable interrupts
- HAL_FDCAN_DeactivateNotification : Disable interrupts
- HAL_FDCAN_IRQHandler : Handles FDCAN interrupt request

This section contains the following APIs:
- ***HAL_FDCAN_ConfigInterruptLines()***
- ***HAL_FDCAN_ActivateNotification()***
- ***HAL_FDCAN_DeactivateNotification()***
- ***HAL_FDCAN_IRQHandler()***

25.2.6 Callback functions

This subsection provides the following callback functions:
- HAL_FDCAN_TxEventFifoCallback
- HAL_FDCAN_RxFifo0Callback
- HAL_FDCAN_RxFifo1Callback
- HAL_FDCAN_TxFifoEmptyCallback
- HAL_FDCAN_TxBufferCompleteCallback
- HAL_FDCAN_TxBufferAbortCallback
- HAL_FDCAN_HighPriorityMessageCallback
- HAL_FDCAN_TimestampWraparoundCallback
- HAL_FDCAN_TimeoutOccurredCallback
- HAL_FDCAN_ErrorCallback
- HAL_FDCAN_ErrorStatusCallback

This section contains the following APIs:
- *HAL_FDCAN_TxEventFifoCallback()*
- *HAL_FDCAN_RxFifo0Callback()*
- *HAL_FDCAN_RxFifo1Callback()*
- *HAL_FDCAN_TxFifoEmptyCallback()*
- *HAL_FDCAN_TxBufferCompleteCallback()*
- *HAL_FDCAN_TxBufferAbortCallback()*
- *HAL_FDCAN_TimestampWraparoundCallback()*
- *HAL_FDCAN_TimeoutOccurredCallback()*
- *HAL_FDCAN_HighPriorityMessageCallback()*
- *HAL_FDCAN_ErrorCallback()*
- *HAL_FDCAN_ErrorStatusCallback()*

25.2.7 Peripheral State functions

This subsection provides functions allowing to :
- HAL_FDCAN_GetState() : Return the FDCAN state.
- HAL_FDCAN_GetError() : Return the FDCAN error code if any.

This section contains the following APIs:
- *HAL_FDCAN_GetState()*
- *HAL_FDCAN_GetError()*

25.2.8 Detailed description of functions

HAL_FDCAN_Init

Function name

HAL_StatusTypeDef HAL_FDCAN_Init (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN peripheral according to the specified parameters in the FDCAN_InitTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DeInit

Function name

HAL_StatusTypeDef HAL_FDCAN_DeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Deinitializes the FDCAN peripheral registers to their default reset values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_MspInit

Function name

void HAL_FDCAN_MspInit (FDCAN_HandleTypeDef * hfdcan)

Function description

Initializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_MspDeInit

Function name

void HAL_FDCAN_MspDeInit (FDCAN_HandleTypeDef * hfdcan)

Function description

DeInitializes the FDCAN MSP.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_EnterPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnterPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enter FDCAN peripheral in sleep mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ExitPowerDownMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitPowerDownMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit power down mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID, pFDCAN_CallbackTypeDef pCallback)

Function description

HAL_FDCAN_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterCallback (FDCAN_HandleTypeDef * hfdcan, HAL_FDCAN_CallbackIDTypeDef CallbackID)

Function description

Unregister a FDCAN CallBack.

Parameters

- **hfdcan:** pointer to a FDCAN_HandleTypeDef structure that contains the configuration information for FDCAN module
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_FDCAN_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty callback ID
  - HAL_FDCAN_HIGH_PRIO_MESSAGE_CB_ID High priority message callback ID
  - HAL_FDCAN_TIMESTAMP_WRAPAROUND_CB_ID Timestamp wraparound callback ID
  - HAL_FDCAN_TIMEOUT_OCCURRED_CB_ID Timeout occurred callback ID
  - HAL_FDCAN_ERROR_CALLBACK_CB_ID Error callback ID
  - HAL_FDCAN_MSPINIT_CB_ID MspInit callback ID
  - HAL_FDCAN_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxEventFifoCallbackTypeDef pCallback)

Function description

Register Tx Event Fifo FDCAN Callback To be used instead of the weak HAL_FDCAN_TxEventFifoCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Event Fifo Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxEventFifoCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Event Fifo FDCAN Callback Tx Event Fifo FDCAN Callback is redirected to the weak HAL_FDCAN_TxEventFifoCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo0CallbackTypeDef pCallback)

Function description

Register Rx Fifo 0 FDCAN Callback To be used instead of the weak HAL_FDCAN_RxFifo0Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Rx Fifo 0 Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterRxFifo0Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo0Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Rx Fifo 0 FDCAN Callback Rx Fifo 0 FDCAN Callback is redirected to the weak HAL_FDCAN_RxFifo0Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_RxFifo1CallbackTypeDef pCallback)

Function description

Register Rx Fifo 1 FDCAN Callback To be used instead of the weak HAL_FDCAN_RxFifo1Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Rx Fifo 1 Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterRxFifo1Callback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterRxFifo1Callback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Rx Fifo 1 FDCAN Callback Rx Fifo 1 FDCAN Callback is redirected to the weak HAL_FDCAN_RxFifo1Callback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferCompleteCallbackTypeDef pCallback)

Function description

Register Tx Buffer Complete FDCAN Callback To be used instead of the weak HAL_FDCAN_TxBufferCompleteCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Buffer Complete Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxBufferCompleteCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Buffer Complete FDCAN Callback Tx Buffer Complete FDCAN Callback is redirected to the weak HAL_FDCAN_TxBufferCompleteCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_TxBufferAbortCallbackTypeDef pCallback)

Function description

Register Tx Buffer Abort FDCAN Callback To be used instead of the weak HAL_FDCAN_TxBufferAbortCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Tx Buffer Abort Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterTxBufferAbortCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterTxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Tx Buffer Abort FDCAN Callback Tx Buffer Abort FDCAN Callback is redirected to the weak HAL_FDCAN_TxBufferAbortCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_RegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_RegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, pFDCAN_ErrorStatusCallbackTypeDef pCallback)

Function description

Register Error Status FDCAN Callback To be used instead of the weak HAL_FDCAN_ErrorStatusCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle
- **pCallback:** pointer to the Error Status Callback function

Return values

- **HAL:** status

HAL_FDCAN_UnRegisterErrorStatusCallback

Function name

HAL_StatusTypeDef HAL_FDCAN_UnRegisterErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

UnRegister the Error Status FDCAN Callback Error Status FDCAN Callback is redirected to the weak HAL_FDCAN_ErrorStatusCallback() predefined callback.

Parameters

- **hfdcan:** FDCAN handle

Return values

- **HAL:** status

HAL_FDCAN_ConfigFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigFilter (FDCAN_HandleTypeDef * hfdcan, const FDCAN_FilterTypeDef * sFilterConfig)

Function description

Configure the FDCAN reception filter according to the specified parameters in the FDCAN_FilterTypeDef structure.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **sFilterConfig:** pointer to an FDCAN_FilterTypeDef structure that contains the filter configuration information

Return values

- **HAL:** status

HAL_FDCAN_ConfigGlobalFilter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigGlobalFilter (FDCAN_HandleTypeDef * hfdcan, uint32_t NonMatchingStd, uint32_t NonMatchingExt, uint32_t RejectRemoteStd, uint32_t RejectRemoteExt)

Function description

Configure the FDCAN global filter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **NonMatchingStd:** Defines how received messages with 11-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  – FDCAN_Non_Matching_Frames.
- **NonMatchingExt:** Defines how received messages with 29-bit IDs that do not match any element of the filter list are treated. This parameter can be a value of
  – FDCAN_Non_Matching_Frames.
- **RejectRemoteStd:** Filter or reject all the remote 11-bit IDs frames. This parameter can be a value of
  – FDCAN_Reject_Remote_Frames.
- **RejectRemoteExt:** Filter or reject all the remote 29-bit IDs frames. This parameter can be a value of
  – FDCAN_Reject_Remote_Frames.

Return values

- **HAL:** status

HAL_FDCAN_ConfigExtendedIdMask

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigExtendedIdMask (FDCAN_HandleTypeDef * hfdcan, uint32_t Mask)

Function description

Configure the extended ID mask.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **Mask:** Extended ID Mask. This parameter must be a number between 0 and 0x1FFFFFFF.

Return values

- **HAL:** status

HAL_FDCAN_ConfigRxFifoOverwrite

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRxFifoOverwrite (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo, uint32_t OperationMode)

Function description

Configure the Rx FIFO operation mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  - FDCAN_RX_FIFO0: Rx FIFO 0
  - FDCAN_RX_FIFO1: Rx FIFO 1
- **OperationMode:** operation mode. This parameter can be a value of
  - FDCAN_Rx_FIFO_operation_mode.

Return values

- **HAL:** status

HAL_FDCAN_ConfigRamWatchdog

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigRamWatchdog (FDCAN_HandleTypeDef * hfdcan, uint32_t CounterStartValue)

Function description

Configure the RAM watchdog.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **CounterStartValue:** Start value of the Message RAM Watchdog Counter, This parameter must be a number between 0x00 and 0xFF, with the reset value of 0x00 the counter is disabled.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampPrescaler)

Function description

Configure the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampPrescaler:** Timestamp Counter Prescaler. This parameter can be a value of
  – FDCAN_Timestamp_Prescaler.

Return values

- **HAL:** status

HAL_FDCAN_EnableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimestampCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimestampOperation)

Function description

Enable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimestampOperation:** Timestamp counter operation. This parameter can be a value of
  – FDCAN_Timestamp.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timestamp counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimestampCounter

Function name

uint16_t HAL_FDCAN_GetTimestampCounter (const FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timestamp counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Timestamp:** counter value

HAL_FDCAN_ResetTimestampCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimestampCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timestamp counter to zero.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTimeoutCounter (FDCAN_HandleTypeDef * hfdcan, uint32_t TimeoutOperation, uint32_t TimeoutPeriod)

Function description

Configure the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TimeoutOperation:** Timeout counter operation. This parameter can be a value of
  – FDCAN_Timeout_Operation.
- **TimeoutPeriod:** Start value of the timeout down-counter. This parameter must be a number between 0x0000 and 0xFFFF

Return values

- **HAL:** status

HAL_FDCAN_EnableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the timeout counter.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_GetTimeoutCounter

Function name

uint16_t HAL_FDCAN_GetTimeoutCounter (const FDCAN_HandleTypeDef * hfdcan)

Function description

Get the timeout counter value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Timeout:** counter value

HAL_FDCAN_ResetTimeoutCounter

Function name

HAL_StatusTypeDef HAL_FDCAN_ResetTimeoutCounter (FDCAN_HandleTypeDef * hfdcan)

Function description

Reset the timeout counter to its start value.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan, uint32_t TdcOffset, uint32_t TdcFilter)

Function description

Configure the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TdcOffset:** Transmitter Delay Compensation Offset. This parameter must be a number between 0x00 and 0x7F.
- **TdcFilter:** Transmitter Delay Compensation Filter Window Length. This parameter must be a number between 0x00 and 0x7F.

Return values

- **HAL:** status

HAL_FDCAN_EnableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableTxDelayCompensation

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableTxDelayCompensation (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable the transmitter delay compensation.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableISOMode

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableISOMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable ISO 11898-1 protocol mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_EnableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_EnableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Enable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_DisableEdgeFiltering

Function name

HAL_StatusTypeDef HAL_FDCAN_DisableEdgeFiltering (FDCAN_HandleTypeDef * hfdcan)

Function description

Disable edge filtering during bus integration.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Start

Function name

HAL_StatusTypeDef HAL_FDCAN_Start (FDCAN_HandleTypeDef * hfdcan)

Function description

Start the FDCAN module.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_Stop

Function name

HAL_StatusTypeDef HAL_FDCAN_Stop (FDCAN_HandleTypeDef * hfdcan)

Function description

Stop the FDCAN module and enable access to configuration registers.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_AddMessageToTxFifoQ

Function name

HAL_StatusTypeDef HAL_FDCAN_AddMessageToTxFifoQ (FDCAN_HandleTypeDef * hfdcan, const FDCAN_TxHeaderTypeDef * pTxHeader, const uint8_t * pTxData)

Function description

Add a message to the Tx FIFO/Queue and activate the corresponding transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxHeader:** pointer to a FDCAN_TxHeaderTypeDef structure.
- **pTxData:** pointer to a buffer containing the payload of the Tx frame.

Return values

- **HAL:** status

HAL_FDCAN_GetLatestTxFifoQRequestBuffer

Function name

uint32_t HAL_FDCAN_GetLatestTxFifoQRequestBuffer (const FDCAN_HandleTypeDef * hfdcan)

Function description

Get Tx buffer index of latest Tx FIFO/Queue request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** buffer index of last Tx FIFO/Queue request
  - Any value of
    - FDCAN_Tx_location if Tx request has been submitted.
  - 0 if no Tx FIFO/Queue request have been submitted.

HAL_FDCAN_AbortTxRequest

Function name

HAL_StatusTypeDef HAL_FDCAN_AbortTxRequest (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndex)

Function description

Abort transmission request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndex:** buffer index. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **HAL:** status

HAL_FDCAN_GetRxMessage

Function name

HAL_StatusTypeDef HAL_FDCAN_GetRxMessage (FDCAN_HandleTypeDef * hfdcan, uint32_t RxLocation, FDCAN_RxHeaderTypeDef * pRxHeader, uint8_t * pRxData)

Function description

Get an FDCAN frame from the Rx FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxLocation:** Location of the received message to be read. This parameter can be a value of
  – FDCAN_Rx_location.
- **pRxHeader:** pointer to a FDCAN_RxHeaderTypeDef structure.
- **pRxData:** pointer to a buffer where the payload of the Rx frame will be stored.

Return values

- **HAL:** status

HAL_FDCAN_GetTxEvent

Function name

HAL_StatusTypeDef HAL_FDCAN_GetTxEvent (FDCAN_HandleTypeDef * hfdcan, FDCAN_TxEventFifoTypeDef * pTxEvent)

Function description

Get an FDCAN Tx event from the Tx Event FIFO zone into the message RAM.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **pTxEvent:** pointer to a FDCAN_TxEventFifoTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetHighPriorityMessageStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetHighPriorityMessageStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_HpMsgStatusTypeDef * HpMsgStatus)

Function description

Get high priority message status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **HpMsgStatus:** pointer to an FDCAN_HpMsgStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetProtocolStatus

Function name

HAL_StatusTypeDef HAL_FDCAN_GetProtocolStatus (const FDCAN_HandleTypeDef * hfdcan, FDCAN_ProtocolStatusTypeDef * ProtocolStatus)

Function description

Get protocol status.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ProtocolStatus:** pointer to an FDCAN_ProtocolStatusTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_GetErrorCounters

Function name

HAL_StatusTypeDef HAL_FDCAN_GetErrorCounters (const FDCAN_HandleTypeDef * hfdcan, FDCAN_ErrorCountersTypeDef * ErrorCounters)

Function description

Get error counter values.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorCounters:** pointer to an FDCAN_ErrorCountersTypeDef structure.

Return values

- **HAL:** status

HAL_FDCAN_IsTxBufferMessagePending

Function name

uint32_t HAL_FDCAN_IsTxBufferMessagePending (const FDCAN_HandleTypeDef * hfdcan, uint32_t TxBufferIndex)

Function description

Check if a transmission request is pending on the selected Tx buffer.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxBufferIndex:** Tx buffer index. This parameter can be any combination of
  – FDCAN_Tx_location.

Return values

- **Status:**
  – 0 : No pending transmission request on TxBufferIndex list.
  – 1 : Pending transmission request on TxBufferIndex.

HAL_FDCAN_GetRxFifoFillLevel

Function name

uint32_t HAL_FDCAN_GetRxFifoFillLevel (const FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo)

Function description

Return Rx FIFO fill level.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo:** Rx FIFO. This parameter can be one of the following values:
  – FDCAN_RX_FIFO0: Rx FIFO 0
  – FDCAN_RX_FIFO1: Rx FIFO 1

Return values

- **Rx:** FIFO fill level.

HAL_FDCAN_GetTxFifoFreeLevel

Function name

uint32_t HAL_FDCAN_GetTxFifoFreeLevel (const FDCAN_HandleTypeDef * hfdcan)

Function description

Return Tx FIFO free level: number of consecutive free Tx FIFO elements starting from Tx FIFO GetIndex.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Tx:** FIFO free level.

HAL_FDCAN_IsRestrictedOperationMode

Function name

uint32_t HAL_FDCAN_IsRestrictedOperationMode (const FDCAN_HandleTypeDef * hfdcan)

Function description

Check if the FDCAN peripheral entered Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **Status:**
  – 0 : Normal FDCAN operation.
  – 1 : Restricted Operation Mode active.

HAL_FDCAN_ExitRestrictedOperationMode

Function name

HAL_StatusTypeDef HAL_FDCAN_ExitRestrictedOperationMode (FDCAN_HandleTypeDef * hfdcan)

Function description

Exit Restricted Operation Mode.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_ConfigInterruptLines

Function name

HAL_StatusTypeDef HAL_FDCAN_ConfigInterruptLines (FDCAN_HandleTypeDef * hfdcan, uint32_t ITList, uint32_t InterruptLine)

Function description

Assign interrupts to either Interrupt line 0 or 1.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ITList:** indicates which interrupts group will be assigned to the selected interrupt line. This parameter can be any combination of
  - FDCAN_Interrupts_Group.
- **InterruptLine:** Interrupt line. This parameter can be a value of
  - FDCAN_Interrupt_Line.

Return values

- **HAL:** status

HAL_FDCAN_ActivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_ActivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t ActiveITs, uint32_t BufferIndexes)

Function description

Enable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ActiveITs:** indicates which interrupts will be enabled. This parameter can be any combination of
  - FDCAN_Interrupts.
- **BufferIndexes:** Tx Buffer Indexes. This parameter can be any combination of
  - FDCAN_Tx_location. This parameter is ignored if ActiveITs does not include one of the following:
    - FDCAN_IT_TX_COMPLETE
    - FDCAN_IT_TX_ABORT_COMPLETE

Return values

- **HAL:** status

HAL_FDCAN_DeactivateNotification

Function name

HAL_StatusTypeDef HAL_FDCAN_DeactivateNotification (FDCAN_HandleTypeDef * hfdcan, uint32_t InactiveITs)

Function description

Disable interrupts.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **InactiveITs:** indicates which interrupts will be disabled. This parameter can be any combination of
  – FDCAN_Interrupts.

Return values

- **HAL:** status

HAL_FDCAN_IRQHandler

Function name

void HAL_FDCAN_IRQHandler (FDCAN_HandleTypeDef * hfdcan)

Function description

Handles FDCAN interrupt request.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** status

HAL_FDCAN_TxEventFifoCallback

Function name

void HAL_FDCAN_TxEventFifoCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t TxEventFifoITs)

Function description

Tx Event callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **TxEventFifoITs:** indicates which Tx Event FIFO interrupts are signaled. This parameter can be any combination of
  – FDCAN_Tx_Event_Fifo_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo0Callback

Function name

void HAL_FDCAN_RxFifo0Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo0ITs)

Function description

Rx FIFO 0 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo0ITs:** indicates which Rx FIFO 0 interrupts are signaled. This parameter can be any combination of
  - FDCAN_Rx_Fifo0_Interrupts.

Return values

- **None:**

HAL_FDCAN_RxFifo1Callback

Function name

void HAL_FDCAN_RxFifo1Callback (FDCAN_HandleTypeDef * hfdcan, uint32_t RxFifo1ITs)

Function description

Rx FIFO 1 callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **RxFifo1ITs:** indicates which Rx FIFO 1 interrupts are signaled. This parameter can be any combination of
  - FDCAN_Rx_Fifo1_Interrupts.

Return values

- **None:**

HAL_FDCAN_TxFifoEmptyCallback

Function name

void HAL_FDCAN_TxFifoEmptyCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Tx FIFO Empty callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TxBufferCompleteCallback

Function name

void HAL_FDCAN_TxBufferCompleteCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Complete callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the transmitted buffers. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_TxBufferAbortCallback

Function name

void HAL_FDCAN_TxBufferAbortCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t BufferIndexes)

Function description

Transmission Cancellation callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **BufferIndexes:** Indexes of the aborted buffers. This parameter can be any combination of
  - FDCAN_Tx_location.

Return values

- **None:**

HAL_FDCAN_HighPriorityMessageCallback

Function name

void HAL_FDCAN_HighPriorityMessageCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

High Priority Message callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimestampWraparoundCallback

Function name

void HAL_FDCAN_TimestampWraparoundCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timestamp Wraparound callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_TimeoutOccurredCallback

Function name

void HAL_FDCAN_TimeoutOccurredCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Timeout Occurred callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorCallback

Function name

void HAL_FDCAN_ErrorCallback (FDCAN_HandleTypeDef * hfdcan)

Function description

Error callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **None:**

HAL_FDCAN_ErrorStatusCallback

Function name

void HAL_FDCAN_ErrorStatusCallback (FDCAN_HandleTypeDef * hfdcan, uint32_t ErrorStatusITs)

Function description

Error status callback.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.
- **ErrorStatusITs:** indicates which Error Status interrupts are signaled. This parameter can be any combination of
  – FDCAN_Error_Status_Interrupts.

Return values

- **None:**

HAL_FDCAN_GetError

Function name

uint32_t HAL_FDCAN_GetError (const FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN error code.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **FDCAN:** Error Code

HAL_FDCAN_GetState

Function name

HAL_FDCAN_StateTypeDef HAL_FDCAN_GetState (const FDCAN_HandleTypeDef * hfdcan)

Function description

Return the FDCAN state.

Parameters

- **hfdcan:** pointer to an FDCAN_HandleTypeDef structure that contains the configuration information for the specified FDCAN.

Return values

- **HAL:** state

25.3 FDCAN Firmware driver defines

The following section lists the various define and macros of the module.

25.3.1 FDCAN

FDCAN

FDCAN Bit Rate Switching

FDCAN_BRS_OFF

FDCAN frames transmitted/received without bit rate switching

FDCAN_BRS_ON

FDCAN frames transmitted/received with bit rate switching

FDCAN Clock Divider

FDCAN_CLOCK_DIV1

Divide kernel clock by 1

FDCAN_CLOCK_DIV2

Divide kernel clock by 2

FDCAN_CLOCK_DIV4

Divide kernel clock by 4

FDCAN_CLOCK_DIV6

Divide kernel clock by 6

FDCAN_CLOCK_DIV8

Divide kernel clock by 8

FDCAN_CLOCK_DIV10

Divide kernel clock by 10

FDCAN_CLOCK_DIV12

Divide kernel clock by 12

FDCAN_CLOCK_DIV14

Divide kernel clock by 14

FDCAN_CLOCK_DIV16

Divide kernel clock by 16

FDCAN_CLOCK_DIV18

Divide kernel clock by 18

FDCAN_CLOCK_DIV20

Divide kernel clock by 20

FDCAN_CLOCK_DIV22

Divide kernel clock by 22

FDCAN_CLOCK_DIV24

Divide kernel clock by 24

FDCAN_CLOCK_DIV26

Divide kernel clock by 26

FDCAN_CLOCK_DIV28

Divide kernel clock by 28

FDCAN_CLOCK_DIV30

Divide kernel clock by 30

FDCAN communication state

FDCAN_COM_STATE_SYNC

Node is synchronizing on CAN communication

FDCAN_COM_STATE_IDLE

Node is neither receiver nor transmitter

FDCAN_COM_STATE_RX

Node is operating as receiver

FDCAN_COM_STATE_TX

Node is operating as transmitter

FDCAN Counter Interrupts

FDCAN_IT_TIMESTAMP_WRAPAROUND

Timestamp counter wrapped around

FDCAN_IT_TIMEOUT_OCCURRED

Timeout reached

FDCAN Data Length Code

FDCAN_DLC_BYTES_0

0 bytes data field

FDCAN_DLC_BYTES_1

1 bytes data field

FDCAN_DLC_BYTES_2

2 bytes data field

FDCAN_DLC_BYTES_3

3 bytes data field

FDCAN_DLC_BYTES_4

4 bytes data field

FDCAN_DLC_BYTES_5

5 bytes data field

FDCAN_DLC_BYTES_6

6 bytes data field

FDCAN_DLC_BYTES_7

7 bytes data field

FDCAN_DLC_BYTES_8

8 bytes data field

FDCAN_DLC_BYTES_12

12 bytes data field

FDCAN_DLC_BYTES_16

16 bytes data field

FDCAN_DLC_BYTES_20

20 bytes data field

FDCAN_DLC_BYTES_24

24 bytes data field

FDCAN_DLC_BYTES_32

32 bytes data field

FDCAN_DLC_BYTES_48

48 bytes data field

FDCAN_DLC_BYTES_64

64 bytes data field

FDCAN Error Interrupts

FDCAN_IT_RAM_ACCESS_FAILURE

Message RAM access failure occurred

FDCAN_IT_ERROR_LOGGING_OVERFLOW

Overflow of FDCAN Error Logging Counter occurred

FDCAN_IT_RAM_WATCHDOG

Message RAM Watchdog event due to missing READY

FDCAN_IT_ARB_PROTOCOL_ERROR

Protocol error in arbitration phase detected

FDCAN_IT_DATA_PROTOCOL_ERROR

Protocol error in data phase detected

FDCAN_IT_RESERVED_ADDRESS_ACCESS

Access to reserved address occurred

FDCAN Error State Indicator

FDCAN_ESI_ACTIVE

Transmitting node is error active

FDCAN_ESI_PASSIVE

Transmitting node is error passive

FDCAN Error Status Interrupts

FDCAN_IT_ERROR_PASSIVE

Error_Passive status changed

FDCAN_IT_ERROR_WARNING

Error_Warning status changed

FDCAN_IT_BUS_OFF

Bus_Off status changed

FDCAN Event Type

FDCAN_TX_EVENT

Tx event

FDCAN_TX_IN_SPITE_OF_ABORT

Transmission in spite of cancellation

FDCAN Exported Macros

__HAL_FDCAN_RESET_HANDLE_STATE

Description:
- Reset FDCAN handle state.

Parameters:
- __HANDLE__: FDCAN handle.

Return value:
- None

__HAL_FDCAN_ENABLE_IT

Description:
- Enable the specified FDCAN interrupts.

Parameters:
- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  – FDCAN_Interrupts

Return value:
- None

__HAL_FDCAN_DISABLE_IT

Description:
- Disable the specified FDCAN interrupts.

Parameters:
- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be any combination of
  – FDCAN_Interrupts

Return value:
- None

__HAL_FDCAN_GET_IT

Description:
- Check whether the specified FDCAN interrupt is set or not.

Parameters:
- __HANDLE__: FDCAN handle.
- __INTERRUPT__: FDCAN interrupt. This parameter can be one of
  - FDCAN_Interrupts

Return value:
- ITStatus

__HAL_FDCAN_CLEAR_IT

Description:
- Clear the specified FDCAN interrupts.

Parameters:
- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the interrupts to clear. This parameter can be any combination of
  - FDCAN_Interrupts

Return value:
- None

__HAL_FDCAN_GET_FLAG

Description:
- Check whether the specified FDCAN flag is set or not.

Parameters:
- __HANDLE__: FDCAN handle.
- __FLAG__: FDCAN flag. This parameter can be one of
  - FDCAN_flags

Return value:
- FlagStatus

__HAL_FDCAN_CLEAR_FLAG

Description:
- Clear the specified FDCAN flags.

Parameters:
- __HANDLE__: FDCAN handle.
- __FLAG__: specifies the flags to clear. This parameter can be any combination of
  - FDCAN_flags

Return value:
- None

__HAL_FDCAN_GET_IT_SOURCE

Description:
- Check if the specified FDCAN interrupt source is enabled or disabled.

Parameters:
- __HANDLE__: FDCAN handle.
- __INTERRUPT__: specifies the FDCAN interrupt source to check. This parameter can be a value of
  - FDCAN_Interrupts

Return value:
- ITStatus

FDCAN Filter Configuration

FDCAN_FILTER_DISABLE

Disable filter element

FDCAN_FILTER_TO_RXFIFO0

Store in Rx FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1

Store in Rx FIFO 1 if filter matches

FDCAN_FILTER_REJECT

Reject ID if filter matches

FDCAN_FILTER_HP

Set high priority if filter matches

FDCAN_FILTER_TO_RXFIFO0_HP

Set high priority and store in FIFO 0 if filter matches

FDCAN_FILTER_TO_RXFIFO1_HP

Set high priority and store in FIFO 1 if filter matches

FDCAN Filter Type

FDCAN_FILTER_RANGE

Range filter from FilterID1 to FilterID2

FDCAN_FILTER_DUAL

Dual ID filter for FilterID1 or FilterID2

FDCAN_FILTER_MASK

Classic filter: FilterID1 = filter, FilterID2 = mask

FDCAN_FILTER_RANGE_NO_EIDM

Range filter from FilterID1 to FilterID2, EIDM mask not applied

FDCAN Flags

FDCAN_FLAG_TX_COMPLETE

Transmission Completed

FDCAN_FLAG_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_FLAG_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN_FLAG_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN_FLAG_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_FLAG_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_FLAG_TX_EVT_FIFO_NEW_DATA
Tx Handler wrote Tx Event FIFO element

FDCAN_FLAG_RX_FIFO0_MESSAGE_LOST
Rx FIFO 0 message lost

FDCAN_FLAG_RX_FIFO0_FULL
Rx FIFO 0 full

FDCAN_FLAG_RX_FIFO0_NEW_MESSAGE
New message written to Rx FIFO 0

FDCAN_FLAG_RX_FIFO1_MESSAGE_LOST
Rx FIFO 1 message lost

FDCAN_FLAG_RX_FIFO1_FULL
Rx FIFO 1 full

FDCAN_FLAG_RX_FIFO1_NEW_MESSAGE
New message written to Rx FIFO 1

FDCAN_FLAG_RAM_ACCESS_FAILURE
Message RAM access failure occurred

FDCAN_FLAG_ERROR_LOGGING_OVERFLOW
Overflow of FDCAN Error Logging Counter occurred

FDCAN_FLAG_ERROR_PASSIVE
Error_Passive status changed

FDCAN_FLAG_ERROR_WARNING
Error_Warning status changed

FDCAN_FLAG_BUS_OFF
Bus_Off status changed

FDCAN_FLAG_RAM_WATCHDOG
Message RAM Watchdog event due to missing READY

FDCAN_FLAG_ARB_PROTOCOL_ERROR
Protocol error in arbitration phase detected

FDCAN_FLAG_DATA_PROTOCOL_ERROR
Protocol error in data phase detected

FDCAN_FLAG_RESERVED_ADDRESS_ACCESS
Access to reserved address occurred

FDCAN_FLAG_TIMESTAMP_WRAPAROUND
Timestamp counter wrapped around

FDCAN_FLAG_TIMEOUT_OCCURRED
Timeout reached

FDCAN format

FDCAN_CLASSIC_CAN
Frame transmitted/received in Classic CAN format

FDCAN_FD_CAN

Frame transmitted/received in FDCAN format

FDCAN Frame Format

FDCAN_FRAME_CLASSIC

Classic mode

FDCAN_FRAME_FD_NO_BRS

FD mode without BitRate Switching

FDCAN_FRAME_FD_BRS

FD mode with BitRate Switching

FDCAN Frame Type

FDCAN_DATA_FRAME

Data frame

FDCAN_REMOTE_FRAME

Remote frame

FDCAN High Priority Message Storage

FDCAN_HP_STORAGE_NO_FIFO

No FIFO selected

FDCAN_HP_STORAGE_MSG_LOST

FIFO message lost

FDCAN_HP_STORAGE_RXFIFO0

Message stored in FIFO 0

FDCAN_HP_STORAGE_RXFIFO1

Message stored in FIFO 1

FDCAN ID Type

FDCAN_STANDARD_ID

Standard ID element

FDCAN_EXTENDED_ID

Extended ID element

FDCAN Interrupts Group

FDCAN_IT_GROUP_RX_FIFO0

RX FIFO 0 Interrupts Group: RF0LL: Rx FIFO 0 Message Lost RF0FL: Rx FIFO 0 is Full RF0NL: Rx FIFO 0 Has New Message

FDCAN_IT_GROUP_RX_FIFO1

RX FIFO 1 Interrupts Group: RF1LL: Rx FIFO 1 Message Lost RF1FL: Rx FIFO 1 is Full RF1NL: Rx FIFO 1 Has New Message

FDCAN_IT_GROUP_SMSG

Status Message Interrupts Group: TCFL: Transmission Cancellation Finished TCL: Transmission Completed HPML: High Priority Message

FDCAN_IT_GROUP_TX_FIFO_ERROR

TX FIFO Error Interrupts Group: TEFLL: Tx Event FIFO Element Lost TEFFL: Tx Event FIFO Full TEFNL: Tx Event FIFO New Entry TFEL: Tx FIFO Empty Interrupt Line

FDCAN_IT_GROUP_MISC

Misc. Interrupts Group: TOOL: Timeout Occurred MRAFL: Message RAM Access Failure TSWL: Timestamp Wraparound

FDCAN_IT_GROUP_BIT_LINE_ERROR

Bit and Line Error Interrupts Group: EPL: Error Passive ELOL: Error Logging Overflow

FDCAN_IT_GROUP_PROTOCOL_ERROR

Protocol Error Group: ARAL: Access to Reserved Address Line PEDL: Protocol Error in Data Phase Line PEAL: Protocol Error in Arbitration Phase Line WDIL: Watchdog Interrupt Line BOL: Bus_Off Status EWL: Warning Status

FDCAN Interrupts List

FDCAN_IT_LIST_RX_FIFO0

RX FIFO 0 Interrupts List

FDCAN_IT_LIST_RX_FIFO1

RX FIFO 1 Interrupts List

FDCAN_IT_LIST_SMSG

Status Message Interrupts List

FDCAN_IT_LIST_TX_FIFO_ERROR

TX FIFO Error Interrupts List

FDCAN_IT_LIST_MISC

Misc. Interrupts List

FDCAN_IT_LIST_BIT_LINE_ERROR

Bit and Line Error Interrupts List

FDCAN_IT_LIST_PROTOCOL_ERROR

Protocol Error Interrupts List

FDCAN interrupt line

FDCAN_INTERRUPT_LINE0

Interrupt Line 0

FDCAN_INTERRUPT_LINE1

Interrupt Line 1

FDCAN non-matching frames

FDCAN_ACCEPT_IN_RX_FIFO0

Accept in Rx FIFO 0

FDCAN_ACCEPT_IN_RX_FIFO1

Accept in Rx FIFO 1

FDCAN_REJECT

Reject

FDCAN Operating Mode

FDCAN_MODE_NORMAL
Normal mode

FDCAN_MODE_RESTRICTED_OPERATION
Restricted Operation mode

FDCAN_MODE_BUS_MONITORING
Bus Monitoring mode

FDCAN_MODE_INTERNAL_LOOPBACK
Internal LoopBack mode

FDCAN_MODE_EXTERNAL_LOOPBACK
External LoopBack mode

FDCAN protocol error code

FDCAN_PROTOCOL_ERROR_NONE
No error occurred

FDCAN_PROTOCOL_ERROR_STUFF
Stuff error

FDCAN_PROTOCOL_ERROR_FORM
Form error

FDCAN_PROTOCOL_ERROR_ACK
Acknowledge error

FDCAN_PROTOCOL_ERROR_BIT1
Bit 1 (recessive) error

FDCAN_PROTOCOL_ERROR_BIT0
Bit 0 (dominant) error

FDCAN_PROTOCOL_ERROR_CRC
CRC check sum error

FDCAN_PROTOCOL_ERROR_NO_CHANGE
No change since last read

FDCAN reject remote frames

FDCAN_FILTER_REMOTE
Filter remote frames

FDCAN_REJECT_REMOTE
Reject all remote frames

FDCAN Rx FIFO 0 Interrupts

FDCAN_IT_RX_FIFO0_MESSAGE_LOST
Rx FIFO 0 message lost

FDCAN_IT_RX_FIFO0_FULL
Rx FIFO 0 full

FDCAN_IT_RX_FIFO0_NEW_MESSAGE

New message written to Rx FIFO 0

FDCAN Rx FIFO 1 Interrupts

FDCAN_IT_RX_FIFO1_MESSAGE_LOST

Rx FIFO 1 message lost

FDCAN_IT_RX_FIFO1_FULL

Rx FIFO 1 full

FDCAN_IT_RX_FIFO1_NEW_MESSAGE

New message written to Rx FIFO 1

FDCAN FIFO operation mode

FDCAN_RX_FIFO_BLOCKING

Rx FIFO blocking mode

FDCAN_RX_FIFO_OVERWRITE

Rx FIFO overwrite mode

FDCAN Rx Interrupts

FDCAN_IT_RX_HIGH_PRIORITY_MSG

High priority message received

FDCAN Rx Location

FDCAN_RX_FIFO0

Get received message from Rx FIFO 0

FDCAN_RX_FIFO1

Get received message from Rx FIFO 1

FDCAN timeout operation

FDCAN_TIMEOUT_CONTINUOUS

Timeout continuous operation

FDCAN_TIMEOUT_TX_EVENT_FIFO

Timeout controlled by Tx Event FIFO

FDCAN_TIMEOUT_RX_FIFO0

Timeout controlled by Rx FIFO 0

FDCAN_TIMEOUT_RX_FIFO1

Timeout controlled by Rx FIFO 1

FDCAN timestamp

FDCAN_TIMESTAMP_INTERNAL

Timestamp counter value incremented according to TCP

FDCAN_TIMESTAMP_EXTERNAL

External timestamp counter value used

FDCAN timestamp prescaler

FDCAN_TIMESTAMP_PRESC_1

Timestamp counter time unit in equal to CAN bit time

FDCAN_TIMESTAMP_PRESC_2

Timestamp counter time unit in equal to CAN bit time multiplied by 2

FDCAN_TIMESTAMP_PRESC_3

Timestamp counter time unit in equal to CAN bit time multiplied by 3

FDCAN_TIMESTAMP_PRESC_4

Timestamp counter time unit in equal to CAN bit time multiplied by 4

FDCAN_TIMESTAMP_PRESC_5

Timestamp counter time unit in equal to CAN bit time multiplied by 5

FDCAN_TIMESTAMP_PRESC_6

Timestamp counter time unit in equal to CAN bit time multiplied by 6

FDCAN_TIMESTAMP_PRESC_7

Timestamp counter time unit in equal to CAN bit time multiplied by 7

FDCAN_TIMESTAMP_PRESC_8

Timestamp counter time unit in equal to CAN bit time multiplied by 8

FDCAN_TIMESTAMP_PRESC_9

Timestamp counter time unit in equal to CAN bit time multiplied by 9

FDCAN_TIMESTAMP_PRESC_10

Timestamp counter time unit in equal to CAN bit time multiplied by 10

FDCAN_TIMESTAMP_PRESC_11

Timestamp counter time unit in equal to CAN bit time multiplied by 11

FDCAN_TIMESTAMP_PRESC_12

Timestamp counter time unit in equal to CAN bit time multiplied by 12

FDCAN_TIMESTAMP_PRESC_13

Timestamp counter time unit in equal to CAN bit time multiplied by 13

FDCAN_TIMESTAMP_PRESC_14

Timestamp counter time unit in equal to CAN bit time multiplied by 14

FDCAN_TIMESTAMP_PRESC_15

Timestamp counter time unit in equal to CAN bit time multiplied by 15

FDCAN_TIMESTAMP_PRESC_16

Timestamp counter time unit in equal to CAN bit time multiplied by 16

FDCAN Tx FIFO/Queue Mode

FDCAN_TX_FIFO_OPERATION

FIFO mode

FDCAN_TX_QUEUE_OPERATION

Queue mode

FDCAN Tx Event FIFO Interrupts

FDCAN_IT_TX_EVT_FIFO_ELT_LOST

Tx Event FIFO element lost

FDCAN_IT_TX_EVT_FIFO_FULL

Tx Event FIFO full

FDCAN_IT_TX_EVT_FIFO_NEW_DATA

Tx Handler wrote Tx Event FIFO element

FDCAN Tx Interrupts

FDCAN_IT_TX_COMPLETE

Transmission Completed

FDCAN_IT_TX_ABORT_COMPLETE

Transmission Cancellation Finished

FDCAN_IT_TX_FIFO_EMPTY

Tx FIFO Empty

FDCAN Tx Location

FDCAN_TX_BUFFER0

Add message to Tx Buffer 0

FDCAN_TX_BUFFER1

Add message to Tx Buffer 1

FDCAN_TX_BUFFER2

Add message to Tx Buffer 2

26 HAL FLASH Generic Driver

26.1 FLASH Firmware driver registers structures

26.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32u5xx_hal_flash.h

Data Fields

- ***uint32_t TypeErase***
- ***uint32_t Banks***
- ***uint32_t Page***
- ***uint32_t NbPages***

Field Documentation

- ***uint32_t FLASH_EraseInitTypeDef::TypeErase***
  Mass erase or page erase. This parameter can be a value of ***FLASH_Type_Erase***
- ***uint32_t FLASH_EraseInitTypeDef::Banks***
  Select bank to erase. This parameter must be a value of ***FLASH_Banks*** (FLASH_BANK_BOTH should be used only for mass erase)
- ***uint32_t FLASH_EraseInitTypeDef::Page***
  Initial Flash page to erase when page erase is disabled This parameter must be a value between 0 and (max number of pages in the bank - 1) (eg : 127 for 2MB dual bank)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  Number of pages to be erased. This parameter must be a value between 1 and (max number of pages in the bank - value of initial page)

26.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32u5xx_hal_flash.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPArea***
- ***uint32_t WRPStartOffset***
- ***uint32_t WRPEndOffset***
- ***FunctionalState WRPLock***
- ***uint32_t RDPLevel***
- ***uint32_t USERType***
- ***uint32_t USERConfig***
- ***uint32_t WMSecConfig***
- ***uint32_t WMSecStartPage***
- ***uint32_t WMSecEndPage***
- ***uint32_t WMHDPEndPage***
- ***uint32_t BootLock***
- ***uint32_t BootAddrConfig***
- ***uint32_t BootAddr***
- ***uint32_t RDPKeyType***
- ***uint32_t RDPKey1***
- ***uint32_t RDPKey2***

Field Documentation

- ***uint32_t FLASH_OBProgramInitTypeDef::OptionType***
  Option byte to be configured. This parameter can be a combination of the values of ***FLASH_OB_Type***

- ***uint32_t FLASH_OBProgramInitTypeDef::WRPArea***
  Write protection area to be programmed (used for OPTIONBYTE_WRP). Only one WRP area could be programmed at the same time. This parameter can be value of ***FLASH_OB_WRP_Area***
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPStartOffset***
  Write protection start offset (used for OPTIONBYTE_WRP). This parameter must be a value between 0 and (max number of pages in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPEndOffset***
  Write protection end offset (used for OPTIONBYTE_WRP). This parameter must be a value between WRPStartOffset and (max number of pages in the bank - 1)
- ***FunctionalState FLASH_OBProgramInitTypeDef::WRPLock***
  Write protection lock (used for OPTIONBYTE_WRP). This parameter can be set to ENABLE or DISABLE
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPLevel***
  Set the read protection level (used for OPTIONBYTE_RDP). This parameter can be a value of ***FLASH_OB_Read_Protection***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERType***
  User option byte(s) to be configured (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::USERConfig***
  Value of the user option byte (used for OPTIONBYTE_USER). This parameter can be a combination of ***FLASH_OB_USER_BOR_LEVEL***, ***FLASH_OB_USER_nRST_STOP***, ***FLASH_OB_USER_nRST_STANDBY***, ***FLASH_OB_USER_nRST_SHUTDOWN***, ***FLASH_OB_USER_SRAM_RST***, ***FLASH_OB_USER_IWDG_SW***, ***FLASH_OB_USER_IWDG_STOP***, ***FLASH_OB_USER_IWDG_STANDBY***, ***FLASH_OB_USER_WWDG_SW***, ***FLASH_OB_USER_SWAP_BANK***, ***FLASH_OB_USER_DUALBANK***, ***FLASH_OB_USER_BKPRAM_RST***, ***FLASH_OB_USER_SRAM3_ECC***, ***FLASH_OB_USER_SRAM2_ECC***, ***FLASH_OB_USER_SRAM2_RST***, ***FLASH_OB_USER_nSWBOOT0***, ***FLASH_OB_USER_nBOOT0***, ***FLASH_OB_USER_PA15_PUPEN***, ***FLASH_OB_USER_IO_VDD_HSLV***, ***FLASH_OB_USER_IO_VDDIO2_HSLV***, ***FLASH_OB_USER_TZEN***
- ***uint32_t FLASH_OBProgramInitTypeDef::WMSecConfig***
  Configuration of the Watermark-based Secure Area (used for OPTIONBYTE_WMSEC). This parameter must be a value of ***FLASH_OB_WMSEC***
- ***uint32_t FLASH_OBProgramInitTypeDef::WMSecStartPage***
  Start page of secure area (used for OPTIONBYTE_WMSEC). This parameter must be a value between 0 and (max number of pages in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::WMSecEndPage***
  End page of secure area (used for OPTIONBYTE_WMSEC). This parameter must be a value between WMSecStartPage and (max number of pages in the bank - 1)
- ***uint32_t FLASH_OBProgramInitTypeDef::WMHDPEndPage***
  End page of the secure hide protection (used for OPTIONBYTE_WMSEC). This parameter must be a value between WMSecStartPage and WMSecEndPage
- ***uint32_t FLASH_OBProgramInitTypeDef::BootLock***
  Configuration of the boot lock (used for OPTIONBYTE_BOOT_LOCK). This parameter must be a value of ***FLASH_OB_BOOT_LOCK***
- ***uint32_t FLASH_OBProgramInitTypeDef::BootAddrConfig***
  Configuration of the Boot address (used for OPTIONBYTE_BOOTADDR). This parameter must be a value of ***FLASH_OB_BOOTADDR***
- ***uint32_t FLASH_OBProgramInitTypeDef::BootAddr***
  Boot address (used for OPTIONBYTE_BOOTADDR). This parameter must be a value between 0x0 and 0xFFFFFF00
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPKeyType***
  Configuration of the RDP OEM keys (used for OPTIONBYTE_RDPKEY). This parameter can be a value of ***FLASH_OB_RDP_Key_Type***
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPKey1***
  Value of the RDP OEM key 1 (used for OPTIONBYTE_RDPKEY)
- ***uint32_t FLASH_OBProgramInitTypeDef::RDPKey2***
  Value of the RDP OEM key 2 (used for OPTIONBYTE_RDPKEY)

26.1.3 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32u5xx_hal_flash.h

Data Fields

- ***HAL_LockTypeDef Lock***
- ***uint32_t ErrorCode***
- ***uint32_t ProcedureOnGoing***
- ***uint32_t Address***
- ***uint32_t Bank***
- ***uint32_t Page***
- ***uint32_t NbPagesToErase***

Field Documentation

- ***HAL_LockTypeDef FLASH_ProcessTypeDef::Lock***
- ***uint32_t FLASH_ProcessTypeDef::ErrorCode***
- ***uint32_t FLASH_ProcessTypeDef::ProcedureOnGoing***
- ***uint32_t FLASH_ProcessTypeDef::Address***
- ***uint32_t FLASH_ProcessTypeDef::Bank***
- ***uint32_t FLASH_ProcessTypeDef::Page***
- ***uint32_t FLASH_ProcessTypeDef::NbPagesToErase***

26.2 FLASH Firmware driver API description

The following section lists the various functions of the FLASH library.

26.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB C-Bus accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface implements the TrustZone security features (TZ) supported by ARM Cortex-M33 core (CM33).

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- TrustZone aware
- Watermark-based area protection including the secure hide area
- Block-based page protection
- Error code correction (ECC) : Data in flash are 137-bits word (9 bits added per quad-word)

26.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32U5xx devices.

1. Flash Memory IO Programming functions:
   – Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   – Program functions: quad-words and burst program (8 quad-words)
   – There are two modes of programming :
     ◦ Polling mode using HAL_FLASH_Program() function
     ◦ Interrupt mode using HAL_FLASH_Program_IT() function

2. Interrupts and flags management functions :
   – Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
   – Callback functions are called when the flash operations are finished : HAL_FLASH_EndOfOperationCallback() when everything is ok, otherwise HAL_FLASH_OperationErrorCallback()
   – Get error flag status by calling HAL_GetError()
3. Option bytes management functions :
   – Lock and Unlock the option bytes using HAL_FLASH_OB_Unlock() and HAL_FLASH_OB_Lock() functions
   – Launch the reload of the option bytes using HAL_FLASH_Launch() function. In this case, a reset is generated

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:
- Set the latency
- Enable/Disable the Flash power-down during low-power run and sleep modes
- Enable/Disable the Flash interrupts
- Monitor the Flash flags status

26.2.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

This section contains the following APIs:
- *HAL_FLASH_Program()*
- *HAL_FLASH_Program_IT()*
- *HAL_FLASH_IRQHandler()*
- *HAL_FLASH_EndOfOperationCallback()*
- *HAL_FLASH_OperationErrorCallback()*

26.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:
- *HAL_FLASH_Unlock()*
- *HAL_FLASH_Lock()*
- *HAL_FLASH_OB_Unlock()*
- *HAL_FLASH_OB_Lock()*
- *HAL_FLASH_OB_Launch()*

26.2.5 Peripheral Errors functions

This subsection permits to get in run-time Errors of the FLASH peripheral.

This section contains the following APIs:
- *HAL_FLASH_GetError()*

26.2.6 Detailed description of functions

HAL_FLASH_Program

Function name

HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint32_t DataAddress)

Function description

Program a quad-word or a burst of 8 quad-words at a specified address.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type
- **Address:** specifies the address to be programmed. This parameter shall be aligned to the Flash word (128 bits)
- **DataAddress:** specifies the address of data to be programmed. This parameter shall be 32-bit aligned

Return values

- **HAL:** Status

HAL_FLASH_Program_IT

Function name

HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint32_t DataAddress)

Function description

Program a quad-word or a burst of 8 quad-words at a specified address with interrupt enabled.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASH Program Type
- **Address:** specifies the address to be programmed. This parameter shall be aligned to the Flash word (128 bits)
- **DataAddress:** specifies the address of data to be programmed. This parameter shall be 32-bit aligned

Return values

- **HAL:** Status

HAL_FLASH_IRQHandler

Function name

void HAL_FLASH_IRQHandler (void)

Function description

Handle FLASH interrupt request.

Return values

- **None:**

HAL_FLASH_EndOfOperationCallback

Function name

void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)

Function description

FLASH end of operation interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Page Erase: Page which has been erased (if 0xFFFFFFFF, it means that all the selected pages have been erased) Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_OperationErrorCallback

Function name

void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)

Function description

FLASH operation error interrupt callback.

Parameters

- **ReturnValue:** The value saved in this parameter depends on the ongoing procedure Mass Erase: Bank number which has been requested to erase Page Erase: Page number which returned an error Program: Address which was selected for data program

Return values

- **None:**

HAL_FLASH_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_Unlock (void)

Function description

Unlock the FLASH control register access.

Return values

- **HAL:** Status

HAL_FLASH_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_Lock (void)

Function description

Lock the FLASH control register access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Unlock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)

Function description

Unlock the FLASH Option Bytes Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Lock

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)

Function description

Lock the FLASH Option Bytes Registers access.

Return values

- **HAL:** Status

HAL_FLASH_OB_Launch

Function name

HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)

Function description

Launch the option byte loading.

Return values

- **HAL:** Status

HAL_FLASH_GetError

Function name

uint32_t HAL_FLASH_GetError (void)

Function description

Get the specific FLASH error flag.

Return values

- **FLASH_ErrorCode:** The returned value can be:
  - HAL_FLASH_ERROR_NONE: No error set
  - HAL_FLASH_ERROR_OP: FLASH Operation error
  - HAL_FLASH_ERROR_PROG: FLASH Programming error
  - HAL_FLASH_ERROR_WRP: FLASH Write protection error
  - HAL_FLASH_ERROR_PGA: FLASH Programming alignment error
  - HAL_FLASH_ERROR_SIZ: FLASH Size error
  - HAL_FLASH_ERROR_PGS: FLASH Programming sequence error
  - HAL_FLASH_ERROR_OPTW: FLASH Option modification error

FLASH_WaitForLastOperation

Function name

HAL_StatusTypeDef FLASH_WaitForLastOperation (uint32_t Timeout)

Function description

Wait for a FLASH operation to complete.

Parameters

- **Timeout:** maximum flash operation timeout

Return values

- **HAL:** Status

26.3 FLASH Firmware driver defines

The following section lists the various define and macros of the module.

26.3.1 FLASH

FLASH

FLASH Banks

FLASH_BANK_1

Bank 1

FLASH_BANK_2

Bank 2

FLASH_BANK_BOTH

Bank1 and Bank2

FLASH Block-Base Attributes

FLASH_BB_SEC

Flash Block-Based Security Attributes

FLASH_BB_PRIV

Flash Block-Based Privilege Attributes

FLASH Error

HAL_FLASH_ERROR_NONE

HAL_FLASH_ERROR_OP

HAL_FLASH_ERROR_PROG

HAL_FLASH_ERROR_WRP

HAL_FLASH_ERROR_PGA

HAL_FLASH_ERROR_SIZ

HAL_FLASH_ERROR_PGS

HAL_FLASH_ERROR_OPTW

HAL_FLASH_ERROR_ECCC

HAL_FLASH_ERROR_ECCD

FLASH Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- __LATENCY__: FLASH Latency This parameter can be one of the following values :
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states
  - FLASH_LATENCY_5: FLASH Five wait states
  - FLASH_LATENCY_6: FLASH Six wait states
  - FLASH_LATENCY_7: FLASH Seven wait states
  - FLASH_LATENCY_8: FLASH Eight wait states
  - FLASH_LATENCY_9: FLASH Nine wait states
  - FLASH_LATENCY_10: FLASH Ten wait states
  - FLASH_LATENCY_11: FLASH Eleven wait states
  - FLASH_LATENCY_12: FLASH Twelve wait states
  - FLASH_LATENCY_13: FLASH Thirteen wait states
  - FLASH_LATENCY_14: FLASH Fourteen wait states
  - FLASH_LATENCY_15: FLASH Fifteen wait states

Return value:

- None

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency This return value can be one of the following values :
  - FLASH_LATENCY_0: FLASH Zero wait state
  - FLASH_LATENCY_1: FLASH One wait state
  - FLASH_LATENCY_2: FLASH Two wait states
  - FLASH_LATENCY_3: FLASH Three wait states
  - FLASH_LATENCY_4: FLASH Four wait states
  - FLASH_LATENCY_5: FLASH Five wait states
  - FLASH_LATENCY_6: FLASH Six wait states
  - FLASH_LATENCY_7: FLASH Seven wait states
  - FLASH_LATENCY_8: FLASH Eight wait states
  - FLASH_LATENCY_9: FLASH Nine wait states
  - FLASH_LATENCY_10: FLASH Ten wait states
  - FLASH_LATENCY_11: FLASH Eleven wait states
  - FLASH_LATENCY_12: FLASH Twelve wait states
  - FLASH_LATENCY_13: FLASH Thirteen wait states
  - FLASH_LATENCY_14: FLASH Fourteen wait states
  - FLASH_LATENCY_15: FLASH Fifteen wait states

__HAL_FLASH_PREFETCH_BUFFER_ENABLE

Description:

- Enable the FLASH prefetch buffer.

Return value:

- None

__HAL_FLASH_PREFETCH_BUFFER_DISABLE

Description:

- Disable the FLASH prefetch buffer.

Return value:

- None

__HAL_FLASH_SLEEP_POWERDOWN_ENABLE

Description:

- Enable the FLASH power down during Low-Power sleep mode.

Return value:

- none

__HAL_FLASH_SLEEP_POWERDOWN_DISABLE

Description:

- Disable the FLASH power down during Low-Power sleep mode.

Return value:

- none

FLASH Flags Definition

FLASH_FLAG_EOP

FLASH End of operation flag

FLASH_FLAG_OPERR

FLASH Operation error flag

FLASH_FLAG_PROGERR

FLASH Programming error flag

FLASH_FLAG_WRPERR

FLASH Write protection error flag

FLASH_FLAG_PGAERR

FLASH Programming alignment error flag

FLASH_FLAG_SIZERR

FLASH Size error flag

FLASH_FLAG_PGSERR

FLASH Programming sequence error flag

FLASH_FLAG_OPTWERR

FLASH Option modification error flag

FLASH_FLAG_BSY

FLASH Busy flag

FLASH_FLAG_WDW

FLASH Wait Data to Write flag

FLASH_FLAG_ECCC

FLASH ECC correction

FLASH_FLAG_ECCD

FLASH ECC detection

FLASH_FLAG_SR_ERRORS

FLASH_FLAG_ECCR_ERRORS

FLASH_FLAG_ALL_ERRORS

FLASH Interrupts Macros

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP: End of FLASH Operation Interrupt
  - FLASH_IT_OPERR: Error Interrupt
  - FLASH_IT_ECCC: ECC Correction Interrupt

Return value:

- none

__HAL_FLASH_ENABLE_IT_NS

__HAL_FLASH_DISABLE_IT

Description:

- Disable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  - FLASH_IT_EOP: End of FLASH Operation Interrupt
  - FLASH_IT_OPERR: Error Interrupt
  - FLASH_IT_ECCC: ECC Correction Interrupt

Return value:

- none

__HAL_FLASH_DISABLE_IT_NS

__HAL_FLASH_GET_FLAG

Description:

- Check whether the specified FLASH flag is set or not.

Parameters:

- __FLAG__: specifies the FLASH flag to check. This parameter can be one of the following values:
  – FLASH_FLAG_EOP: FLASH End of Operation flag
  – FLASH_FLAG_OPERR: FLASH Operation error flag
  – FLASH_FLAG_PROGERR: FLASH Programming error flag
  – FLASH_FLAG_WRPERR: FLASH Write protection error flag
  – FLASH_FLAG_PGAERR: FLASH Programming alignment error flag
  – FLASH_FLAG_SIZERR: FLASH Size error flag
  – FLASH_FLAG_PGSERR: FLASH Programming sequence error flag
  – FLASH_FLAG_OPTWERR: FLASH Option modification error flag
  – FLASH_FLAG_BSY: FLASH write/erase operations in progress flag
  – FLASH_FLAG_WDW: FLASH Wait Data to Write flag
  – FLASH_FLAG_ECCC: FLASH one ECC error has been detected and corrected
  – FLASH_FLAG_ECCD: FLASH two ECC errors have been detected

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

__HAL_FLASH_GET_FLAG_NS

__HAL_FLASH_CLEAR_FLAG

Description:

- Clear the FLASH's pending flags.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  – FLASH_FLAG_EOP: FLASH End of Operation flag
  – FLASH_FLAG_OPERR: FLASH Operation error flag
  – FLASH_FLAG_PROGERR: FLASH Programming error flag
  – FLASH_FLAG_WRPERR: FLASH Write protection error flag
  – FLASH_FLAG_PGAERR: FLASH Programming alignment error flag
  – FLASH_FLAG_SIZERR: FLASH Size error flag
  – FLASH_FLAG_PGSERR: FLASH Programming sequence error flag
  – FLASH_FLAG_OPTWERR: FLASH Option modification error flag (Only in non-secure)
  – FLASH_FLAG_ECCC: FLASH one ECC error has been detected and corrected
  – FLASH_FLAG_ECCD: FLASH two ECC errors have been detected
  – FLASH_FLAG_ALL_ERRORS: FLASH All errors flags

Return value:

- None

__HAL_FLASH_CLEAR_FLAG_NS

FLASH Interrupts Definition

FLASH_IT_EOP

End of FLASH Operation Interrupt source

FLASH_IT_OPERR

Error Interrupt source

FLASH_IT_ECCC

ECC Correction Interrupt source

FLASH Keys

FLASH_KEY1

Flash key1

FLASH_KEY2

Flash key2: used with FLASH_KEY1 to unlock the FLASH registers access

FLASH_PDKEY1_1

Flash Bank 1 power down key1

FLASH_PDKEY1_2

Flash Bank 1 power down key2: used with FLASH_PDKEY1_1 to unlock the PDREQ1 bit in FLASH_ACR

FLASH_PDKEY2_1

Flash Bank 2 power down key1

FLASH_PDKEY2_2

Flash Bank 2 power down key2: used with FLASH_PDKEY2_1 to unlock the PDREQ2 bit in FLASH_ACR

FLASH_OPTKEY1

Flash option byte key1

FLASH_OPTKEY2

Flash option byte key2: used with FLASH_OPTKEY1 to allow option bytes operations

FLASH Latency

FLASH_LATENCY_0

FLASH Zero wait state

FLASH_LATENCY_1

FLASH One wait state

FLASH_LATENCY_2

FLASH Two wait states

FLASH_LATENCY_3

FLASH Three wait states

FLASH_LATENCY_4

FLASH Four wait states

FLASH_LATENCY_5

FLASH Five wait state

FLASH_LATENCY_6

FLASH Six wait state

FLASH_LATENCY_7

FLASH Seven wait states

FLASH_LATENCY_8

FLASH Eight wait states

FLASH_LATENCY_9

FLASH Nine wait states

FLASH_LATENCY_10

FLASH Ten wait state

FLASH_LATENCY_11

FLASH Eleven wait state

FLASH_LATENCY_12

FLASH Twelve wait states

FLASH_LATENCY_13

FLASH Thirteen wait states

FLASH_LATENCY_14

FLASH Fourteen wait states

FLASH_LATENCY_15

FLASH Fifteen wait states

FLASH LPM configuration

FLASH_LPM_DISABLE

Flash is in normal read mode

FLASH_LPM_ENABLE

Flash is in low-power read mode

FLASH Option Bytes Boot address

OB_BOOTADDR_NS0

Non-secure boot address 0

OB_BOOTADDR_NS1

Non-secure boot address 1

OB_BOOTADDR_SEC0

Secure boot address 0

FLASH Option Bytes Boot Lock

OB_BOOT_LOCK_DISABLE

Boot lock disable

OB_BOOT_LOCK_ENABLE

Boot lock enable

FLASH Option Bytes Read Protection Key Type

OB_RDP_KEY_OEM1

OEM1 key

OB_RDP_KEY_OEM2

OEM2 key

FLASH Option Bytes Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_0_5

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2

FLASH Option Bytes Type

OPTIONBYTE_WRP

WRP option byte configuration

OPTIONBYTE_RDP

RDP option byte configuration

OPTIONBYTE_USER

USER option byte configuration

OPTIONBYTE_BOOTADDR

Boot address option byte configuration

OPTIONBYTE_WMSEC

Watermark-based secure area option byte configuration

OPTIONBYTE_BOOT_LOCK

Boot lock option byte configuration

OPTIONBYTE_RDPKEY

RDP Key option byte configuration

FLASH Option Bytes User BKPRAM ECC check

OB_BKPRAM_ECC_ENABLE

BKPRAM ECC check enable

OB_BKPRAM_ECC_DISABLE

BKPRAM ECC check disable

FLASH Option Bytes User BOR Level

OB_BOR_LEVEL_0

Reset level threshold is around 1.7V

OB_BOR_LEVEL_1

Reset level threshold is around 2.0V

OB_BOR_LEVEL_2

Reset level threshold is around 2.2V

OB_BOR_LEVEL_3

Reset level threshold is around 2.5V

OB_BOR_LEVEL_4

Reset level threshold is around 2.8V

FLASH Option Bytes User Dual-bank Type

OB_DUALBANK_SINGLE

1MB/512KB Single-bank Flash

OB_DUALBANK_DUAL

1MB/512KB Dual-bank Flash

FLASH Option Bytes User High speed IO at low VDDIO2 voltage configuration bit

OB_IO_VDDIO2_HSLV_DISABLE

High-speed IO at low VDDIO2 voltage feature disabled (VDDIO2 can exceed 2.5 V)

OB_IO_VDDIO2_HSLV_ENABLE

High-speed IO at low VDDIO2 voltage feature enabled (VDDIO2 remains below 2.5 V)

FLASH Option Bytes User High speed IO at low voltage configuration bit

OB_IO_VDD_HSLV_DISABLE

High-speed IO at low VDD voltage feature disabled (VDD can exceed 2.5 V)

OB_IO_VDD_HSLV_ENABLE

High-speed IO at low VDD voltage feature enabled (VDD remains below 2.5 V)

FLASH Option Bytes User IWDG Mode On Standby

OB_IWDG_STDBY_FREEZE

Independent watchdog counter is frozen in Standby mode

OB_IWDG_STDBY_RUN

Independent watchdog counter is running in Standby mode

FLASH Option Bytes User IWDG Mode On Stop

OB_IWDG_STOP_FREEZE

Independent watchdog counter is frozen in Stop mode

OB_IWDG_STOP_RUN

Independent watchdog counter is running in Stop mode

FLASH Option Bytes User IWDG Type

OB_IWDG_HW

Hardware independent watchdog

OB_IWDG_SW

Software independent watchdog

FLASH Option Bytes User nBOOT0 option bit

OB_NBOOT0_RESET

nBOOT0 = 0

OB_NBOOT0_SET

nBOOT0 = 1

FLASH Option Bytes User Reset On Shutdown

OB_SHUTDOWN_RST

Reset generated when entering the shutdown mode

OB_SHUTDOWN_NORST

No reset generated when entering the shutdown mode

FLASH Option Bytes User Reset On Standby

OB_STANDBY_RST

Reset generated when entering the standby mode

OB_STANDBY_NORST

No reset generated when entering the standby mode

FLASH Option Bytes User Reset On Stop

OB_STOP_RST

Reset generated when entering the stop mode

OB_STOP_NORST

No reset generated when entering the stop mode

FLASH Option Bytes User Software BOOT0

OB_BOOT0_FROM_OB

BOOT0 taken from the option bit nBOOT0

OB_BOOT0_FROM_PIN

BOOT0 taken from BOOT0 pin

FLASH Option Bytes User PA15 pull-up enable option bit

OB_PA15_PUP_DISABLE

USB power delivery dead-battery enabled / TDI pull-up deactivated

OB_PA15_PUP_ENABLE

USB power delivery dead-battery disabled / TDI pull-up activated

FLASH Option Bytes User SRAM2 ECC check

OB_SRAM2_ECC_ENABLE

SRAM2 ECC check enable

OB_SRAM2_ECC_DISABLE

SRAM2 ECC check disable

FLASH Option Bytes User SRAM2 Erase On Reset Type

OB_SRAM2_RST_ERASE

SRAM2 erased when a system reset occurs

OB_SRAM2_RST_NOT_ERASE

SRAM2 is not erased when a system reset occurs

FLASH Option Bytes User SRAM3 ECC check

OB_SRAM3_ECC_ENABLE

SRAM3 ECC check enable

OB_SRAM3_ECC_DISABLE

SRAM3 ECC check disable

FLASH Option Bytes User SRAM Erase On Reset Type

OB_SRAM_RST_ERASE

All SRAMs (except SRAM2 and BKPSRAM) erased when a system reset occurs

OB_SRAM_RST_NOT_ERASE

All SRAMs (except SRAM2 and BKPSRAM) not erased when a system reset occurs

FLASH Option Bytes User Swap banks

OB_SWAP_BANK_DISABLE

Bank 1 is located at address offset 0x0, Bank 2 is located at 0x100000

OB_SWAP_BANK_ENABLE

Bank 1 is located at address offset 0x100000, Bank 2 is located at 0x0

FLASH Option Bytes User Type

OB_USER_BOR_LEV

BOR reset Level

OB_USER_NRST_STOP

Reset generated when entering the stop mode

OB_USER_NRST_STDBY

Reset generated when entering the standby mode

OB_USER_NRST_SHDW

Reset generated when entering the shutdown mode

OB_USER_SRAM_RST

All SRAMs (except SRAM2 and BKPSRAM) erase upon system reset

OB_USER_IWDG_SW

Independent watchdog selection

OB_USER_IWDG_STOP

Independent watchdog counter freeze in stop mode

OB_USER_IWDG_STDBY

Independent watchdog counter freeze in standby mode

OB_USER_WWDG_SW

Window watchdog selection

OB_USER_SWAP_BANK

Swap banks

OB_USER_DUALBANK

Dual-Bank on 1MB/512kB Flash memory devices

OB_USER_BKPRAM_ECC

Backup RAM ECC detection and correction enable

OB_USER_SRAM3_ECC

SRAM3 ECC detection and correction enable

OB_USER_SRAM2_ECC
SRAM2 ECC detection and correction enable

OB_USER_SRAM2_RST
SRAM2 Erase when system reset

OB_USER_NSWBOOT0
Software BOOT0

OB_USER_NBOOT0
nBOOT0 option bit

OB_USER_PA15_PUPEN
PA15 pull-up enable option bit

OB_USER_IO_VDD_HSLV
High speed IO at low voltage configuration bit

OB_USER_IO_VDDIO2_HSLV
High speed IO at low VDDIO2 voltage configuration bit

OB_USER_TZEN
Global TrustZone security enable

FLASH Option Bytes User Global TrustZone

OB_TZEN_DISABLE
Global TrustZone security disabled

OB_TZEN_ENABLE
Global TrustZone security enabled

FLASH Option Bytes User WWDG Type

OB_WWDG_HW
Hardware window watchdog

OB_WWDG_SW
Software window watchdog

FLASH Option Bytes Watermarked-based security configuration

OB_WMSEC_AREA1
Watermarked-based security area for bank 1

OB_WMSEC_AREA2
Watermarked-based security area for bank 2

OB_WMSEC_SECURE_AREA_CONFIG
Configure Watermarked-based security area

OB_WMSEC_HDP_AREA_CONFIG
Configure Watermarked-based secure hide area

OB_WMSEC_HDP_AREA_ENABLE
Enable Watermarked-based secure hide area

OB_WMSEC_HDP_AREA_DISABLE

Disable Watermarked-based secure hide area

FLASH WRP Area

OB_WRPAREA_BANK1_AREAA

Flash Bank 1 Area A

OB_WRPAREA_BANK1_AREAB

Flash Bank 1 Area B

OB_WRPAREA_BANK2_AREAA

Flash Bank 2 Area A

OB_WRPAREA_BANK2_AREAB

Flash Bank 2 Area B

FLASH Operation Area

FLASH_OPERATION_AREA_BANK_1

Operation in Bank 1

FLASH_OPERATION_AREA_BANK_2

Operation in Bank 2

FLASH_OPERATION_AREA_SYSF

Operation in System Flash memory

FLASH Operation Type

FLASH_OPERATION_TYPE_NONE

No Flash operation

FLASH_OPERATION_TYPE_QUADWORD

Single write operation

FLASH_OPERATION_TYPE_BURST

Burst write operation

FLASH_OPERATION_TYPE_PAGEERASE

Page erase operation

FLASH_OPERATION_TYPE_BANKERASE

Bank erase operation

FLASH_OPERATION_TYPE_MASSERASE

Mass erase operation

FLASH_OPERATION_TYPE_OPTIONCHANGE

Option change operation

FLASH Erase Type

FLASH_TYPEERASE_PAGES

Secure pages erase activation

FLASH_TYPEERASE_PAGES_NS

Non-secure pages erase activation

FLASH_TYPEERASE_MASSERASE

Secure flash mass erase activation

FLASH_TYPEERASE_MASSERASE_NS

Non-secure flash mass erase activation

FLASH Program Type

FLASH_TYPEPROGRAM_QUADWORD

Program a quad-word (128-bit) at a specified secure address

FLASH_TYPEPROGRAM_QUADWORD_NS

Program a quad-word (128-bit) at a specified non-secure address

FLASH_TYPEPROGRAM_BURST

Program a burst (8xquad-word) at a specified secure address

FLASH_TYPEPROGRAM_BURST_NS

Program a burst (8xquad-word) at a specified non-secure address

27 HAL FLASH Extension Driver

27.1 FLASHEx Firmware driver registers structures

27.1.1 FLASH_BBAttributesTypeDef

FLASH_BBAttributesTypeDef is defined in the stm32u5xx_hal_flash_ex.h

Data Fields

- ***uint32_t Bank***
- ***uint32_t BBAttributesType***
- ***uint32_t BBAttributes_array***

Field Documentation

- ***uint32_t FLASH_BBAttributesTypeDef::Bank***
  Selection of the associated bank of Block-based Area. This parameter must be a value of ***FLASH_Banks***
- ***uint32_t FLASH_BBAttributesTypeDef::BBAttributesType***
  Block-Based Attributes type. This parameter must be a value of ***FLASH_BB_Attributes***
- ***uint32_t FLASH_BBAttributesTypeDef::BBAttributes_array[FLASH_BLOCKBASED_NB_REG]***
  Each bit specifies the block-based attribute configuration of a page: 0 means page non-protected, 1 means page protected. Protection (secure or privilege) depends on BBAttributesType value

27.1.2 FLASH_OperationTypeDef

FLASH_OperationTypeDef is defined in the stm32u5xx_hal_flash_ex.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t FlashArea***
- ***uint32_t Address***

Field Documentation

- ***uint32_t FLASH_OperationTypeDef::OperationType***
  Flash operation Type. This parameter must be a value of ***FLASH_Operation_Type***
- ***uint32_t FLASH_OperationTypeDef::FlashArea***
  Flash operation memory area. This parameter must be a value of ***FLASH_Operation_Area***
- ***uint32_t FLASH_OperationTypeDef::Address***
  Flash operation Address offset. This parameter is given by bank, and must be a value between 0x0 and 0xFFFF0

27.2 FLASHEx Firmware driver API description

The following section lists the various functions of the FLASHEx library.

27.2.1 Flash Extended features

Comparing to other previous devices, the FLASH interface for STM32U5xx devices contains the following additional features

- Capacity up to 2 Mbyte with dual bank architecture supporting read-while-write capability (RWW)
- Dual bank memory organization
- Watermark-based secure area including the secure hide areas
- Block-based secure pages

27.2.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32U5xx devices. It includes:

1. Flash Memory Erase functions:
   – Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   – Erase function: page Erase and Bank/Mass Erase
   – There are two modes of erase :
     ◦ Polling Mode using HAL_FLASHEx_Erase()
     ◦ Interrupt Mode using HAL_FLASHEx_Erase_IT()
2. Option Bytes Programming function: Use HAL_FLASHEx_OBProgram() to:
   – Configure the write protection for each area
   – Set the Read protection Level
   – Program the user Option Bytes
   – Configure the watermark security for each area including the secure hide areas
   – Configure the boot lock (BOOT_LOCK)
   – Configure the Boot addresses
3. Get Option Bytes Configuration function: Use HAL_FLASHEx_OBGetConfig() to:
   – Get the value of a write protection area
   – Know if the read protection is activated
   – Get the value of the user Option Bytes
   – Get the configuration of a watermark security area including the secure hide areas
   – Get the boot lock (BOOT_LOCK) configuration
   – Get the value of a boot address
4. Block-based secure / privilege area configuration function: Use HAL_FLASHEx_ConfigBBAttributes()
   – Bit-field allowing to secure or un-secure each page
   – Bit-field allowing to privilege or un-privilege each page
5. Get the block-based secure / privilege area configuration function: Use HAL_FLASHEx_GetBBSec()
   – Return the configuration of the block-based security and privilege for all the pages
6. Activation of the secure hide area function: Use HAL_FLASHEx_EnableSecHideProtection()
   – Deny the access to the secure hide area
7. Privilege mode configuration function: Use HAL_FLASHEx_ConfigPrivMode()
   – FLASH register can be protected against non-privilege accesses
8. Get the privilege mode configuration function: Use HAL_FLASHEx_GetPrivMode()
   – Return if the FLASH registers are protected against non-privilege accesses
9. Security inversion configuration function: Use HAL_FLASHEx_ConfigSecInversion()
   – FLASH secure state can be override
10. Get the security inversion configuration function: Use HAL_FLASHEx_GetSecInversion()
    – Return if FLASH secure state is override
11. Enable bank low-power mode function: Use HAL_FLASHEx_EnablePowerDown()
    – Enable low-power mode for Flash Bank 1 and/or Bank 2
12. Enable low-power read mode function: Use HAL_FLASHEx_ConfigLowPowerRead()
    – Enable low-power read mode for Flash memory
13. Get the low-power read mode configuration function: Use HAL_FLASHEx_GetLowPowerRead()
    – Return if FLASH is in low-power read mode or normal read mode
14. Get Flash operation function: Use HAL_FLASHEx_GetOperation()
    – Return information about the on-going Flash operation. After a system reset, return information about the interrupted Flash operation, if any.

27.2.3 Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extended FLASH programming operations Operations.

This section contains the following APIs:

- *HAL_FLASHEx_Erase()*
- *HAL_FLASHEx_Erase_IT()*
- *HAL_FLASHEx_OBProgram()*
- *HAL_FLASHEx_OBGetConfig()*
- *HAL_FLASHEx_ConfigBBAttributes()*
- *HAL_FLASHEx_GetConfigBBAttributes()*
- *HAL_FLASHEx_EnableSecHideProtection()*

27.2.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)

Function description

Perform a mass erase or erase the specified FLASH memory pages.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **PageError:** pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased).

Return values

- **HAL:** Status

HAL_FLASHEx_Erase_IT

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)

Function description

Perform a mass erase or erase the specified FLASH memory pages with interrupt enabled.

Parameters

- **pEraseInit:** pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.

Return values

- **HAL:** Status

HAL_FLASHEx_OBProgram

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Program Option bytes.

Parameters

- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.

Return values

- **HAL:** Status

Notes

- To configure any option bytes, the option lock bit OPTLOCK must be cleared with the call of HAL_FLASH_OB_Unlock() function.
- New option bytes configuration will be taken into account in two cases: after an option bytes launch through the call of HAL_FLASH_OB_Launch()after a power reset (BOR reset or exit from Standby/ Shutdown modes)

HAL_FLASHEx_OBGetConfig

Function name

void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Get the Option bytes configuration.

Parameters

- **pOBInit:** pointer to an FLASH_OBInitStruct structure that contains the configuration information.

Return values

- **None:**

Notes

- The fields pOBInit->WRPArea, pOBInit->WMSecConfig and pOBInit->BootAddrConfig should indicate which area/address is requested for the WRP, WM Security or Boot Address, else no information will be returned

HAL_FLASHEx_ConfigBBAttributes

Function name

HAL_StatusTypeDef HAL_FLASHEx_ConfigBBAttributes (FLASH_BBAttributesTypeDef * pBBAttributes)

Function description

Configure the block-based secure area.

Parameters

- **pBBAttributes:** pointer to an FLASH_BBAttributesTypeDef structure that contains the configuration information for the programming.

Return values

- **HAL:** Status

Notes

- The field pBBAttributes->Bank should indicate which area is requested for the block-based attributes.
- The field pBBAttributes->BBAttributesType should indicate which block-base attribute type is requested: Secure or Privilege.

HAL_FLASHEx_GetConfigBBAttributes

Function name

void HAL_FLASHEx_GetConfigBBAttributes (FLASH_BBAttributesTypeDef * pBBAttributes)

Function description

Return the block-based attributes.

Parameters

- **pBBAttributes:** [in/out] pointer to an FLASH_BBAttributesTypeDef structure that contains the configuration information.

Return values

- **None:**

Notes

- The field pBBAttributes->Bank should indicate which area is requested for the block-based attributes.
- The field pBBAttributes->BBAttributesType should indicate which block-base attribute type is requested: Secure or Privilege.

HAL_FLASHEx_EnableSecHideProtection

Function name

void HAL_FLASHEx_EnableSecHideProtection (uint32_t Banks)

Function description

Activation of the protection of the secure hide area.

Parameters

- **Banks:** indicate the bank concerned by the activation This parameter can be one of the following values:
  – FLASH_BANK_1: Bank1 to be protected
  – FLASH_BANK_2: Bank2 to be protected
  – FLASH_BANK_BOTH: Bank1 and Bank2 to be protected

Return values

- **None:**

HAL_FLASHEx_ConfigPrivMode

Function name

void HAL_FLASHEx_ConfigPrivMode (uint32_t PrivMode)

Function description

Configuration of the privilege attribute.

Parameters

- **PrivMode:** indicate privilege mode configuration This parameter can be one of the following values:
  – FLASH_SPRIV_GRANTED: access to secure Flash registers is granted to privileged or unprivileged access
  – FLASH_SPRIV_DENIED: access to secure Flash registers is denied to unprivileged access
  – FLASH_NSPRIV_GRANTED: access to non-secure Flash registers is granted to privileged or unprivileged access
  – FLASH_NSPRIV_DENIED: access to non-secure Flash registers is denied to unprivilege access

Return values

- **None:**

HAL_FLASHEx_GetPrivMode

Function name

uint32_t HAL_FLASHEx_GetPrivMode (void)

Function description

Return the value of the privilege attribute.

Return values

- **It:** indicates the privilege mode configuration. This return value can be one of the following values:
  - FLASH_SPRIV_GRANTED: access to secure Flash registers is granted to privileged or unprivileged access
  - FLASH_SPRIV_DENIED: access to secure Flash registers is denied to unprivileged access
  - FLASH_NSPRIV_GRANTED: access to non-secure Flash registers is granted to privileged or unprivileged access
  - FLASH_NSPRIV_DENIED: access to Flash registers is denied to unprivilege accessP

HAL_FLASHEx_ConfigSecInversion

Function name

HAL_StatusTypeDef HAL_FLASHEx_ConfigSecInversion (uint32_t SecInvState)

Function description

Configuration of the security inversion.

Parameters

- **SecInvState:** indicate the flash security state configuration This parameter can be one of the following values:
  - FLASH_SEC_INV_DISABLE: Security state of Flash is not inverted
  - FLASH_SEC_INV_ENABLE: Security state of Flash is inverted

Return values

- **HAL:** Status

HAL_FLASHEx_GetSecInversion

Function name

uint32_t HAL_FLASHEx_GetSecInversion (void)

Function description

Return the value of the security inversion.

Return values

- **It:** indicates the flash security state configuration This return value can be one of the following values:
  - FLASH_SEC_INV_DISABLE: Security state of Flash is not inverted
  - FLASH_SEC_INV_ENABLE: Security state of Flash is inverted

HAL_FLASHEx_EnablePowerDown

Function name

HAL_StatusTypeDef HAL_FLASHEx_EnablePowerDown (uint32_t Banks)

Function description

Enable the Power-down Mode for Flash Banks.

Parameters

- **Banks:** indicate which bank to put in power-down mode This parameter can be one of the following values:
  - FLASH_BANK_1: Flash Bank 1
  - FLASH_BANK_2: Flash Bank 2
  - FLASH_BANK_BOTH: Flash Bank 1 and Bank 2

Return values

- **HAL:** Status

HAL_FLASHEx_ConfigLowPowerRead

Function name

HAL_StatusTypeDef HAL_FLASHEx_ConfigLowPowerRead (uint32_t ConfigLPM)

Function description

Configuration of the Low-Power read Mode.

Parameters

- **ConfigLPM:** indicate the Low-Power read Mode configuration. This parameter can be one of the following values:
  – FLASH_LPM_ENABLE: Flash is in low-power read mode
  – FLASH_LPM_DISABLE: Flash is in normal read mode

Return values

- **HAL:** Status

HAL_FLASHEx_GetLowPowerRead

Function name

uint32_t HAL_FLASHEx_GetLowPowerRead (void)

Function description

Return the value of the Low-Power read Mode.

Return values

- **It:** indicates the flash low-power read mode configuration This return value can be one of the following values:
  – FLASH_LPM_ENABLE: Flash is in low-power read mode
  – FLASH_LPM_DISABLE: Flash is in normal read mode

HAL_FLASHEx_GetOperation

Function name

void HAL_FLASHEx_GetOperation (FLASH_OperationTypeDef * pFlashOperation)

Function description

Return the on-going Flash Operation.

Parameters

- **pFlashOperation:** [out] pointer to a FLASH_OperationTypeDef structure that contains the Flash operation information.

Return values

- **None:**

FLASH_PageErase

Function name

void FLASH_PageErase (uint32_t Page, uint32_t Banks)

Function description

Erase the specified FLASH memory page.

Parameters

- **Page:** FLASH page to erase This parameter must be a value between 0 and (max number of pages in the bank - 1)
- **Banks:** Bank(s) where the page will be erased This parameter can be one of the following values:
  - FLASH_BANK_1: Page in bank 1 to be erased
  - FLASH_BANK_2: Page in bank 2 to be erased

Return values

- **None:**

27.3 FLASHEx Firmware driver defines

The following section lists the various define and macros of the module.

27.3.1 FLASHEx

FLASHEx

28 HAL FMAC Generic Driver

28.1 FMAC Firmware driver registers structures

28.1.1 __FMAC_HandleTypeDef

__FMAC_HandleTypeDef is defined in the stm32u5xx_hal_fmac.h

Data Fields

- ***FMAC_TypeDef * Instance***
- ***uint32_t FilterParam***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***int16_t * pInput***
- ***uint16_t InputCurrentSize***
- ***uint16_t * pInputSize***
- ***int16_t * pOutput***
- ***uint16_t OutputCurrentSize***
- ***uint16_t * pOutputSize***
- ***DMA_HandleTypeDef * hdmaIn***
- ***DMA_HandleTypeDef * hdmaOut***
- ***DMA_HandleTypeDef * hdmaPreload***
- ***void(* ErrorCallback***
- ***void(* HalfGetDataCallback***
- ***void(* GetDataCallback***
- ***void(* HalfOutputDataReadyCallback***
- ***void(* OutputDataReadyCallback***
- ***void(* FilterConfigCallback***
- ***void(* FilterPreloadCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_FMAC_StateTypeDef State***
- ***__IO HAL_FMAC_StateTypeDef RdState***
- ***__IO HAL_FMAC_StateTypeDef WrState***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***FMAC_TypeDef* __FMAC_HandleTypeDef::Instance***
  Register base address
- ***uint32_t __FMAC_HandleTypeDef::FilterParam***
  Filter configuration (operation and parameters). Set to 0 if no valid configuration was applied.
- ***uint8_t __FMAC_HandleTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint8_t __FMAC_HandleTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***int16_t* __FMAC_HandleTypeDef::pInput***
  Pointer to FMAC input data buffer
- ***uint16_t __FMAC_HandleTypeDef::InputCurrentSize***
  Number of the input elements already written into FMAC

- ***uint16_t* __FMAC_HandleTypeDef::pInputSize***
  Number of input elements to write (memory allocated to pInput). In case of early interruption of the filter operation, its value will be updated.
- ***int16_t* __FMAC_HandleTypeDef::pOutput***
  Pointer to FMAC output data buffer
- ***uint16_t __FMAC_HandleTypeDef::OutputCurrentSize***
  Number of the output elements already read from FMAC
- ***uint16_t* __FMAC_HandleTypeDef::pOutputSize***
  Number of output elements to read (memory allocated to pOutput). In case of early interruption of the filter operation, its value will be updated.
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaIn***
  FMAC peripheral input data DMA handle parameters
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaOut***
  FMAC peripheral output data DMA handle parameters
- ***DMA_HandleTypeDef* __FMAC_HandleTypeDef::hdmaPreload***
  FMAC peripheral preloaded data (X1, X2 and Y) DMA handle parameters
- ***void(* __FMAC_HandleTypeDef::ErrorCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC error callback
- ***void(* __FMAC_HandleTypeDef::HalfGetDataCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC get half data callback
- ***void(* __FMAC_HandleTypeDef::GetDataCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC get data callback
- ***void(* __FMAC_HandleTypeDef::HalfOutputDataReadyCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC half output data ready callback
- ***void(* __FMAC_HandleTypeDef::OutputDataReadyCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC output data ready callback
- ***void(* __FMAC_HandleTypeDef::FilterConfigCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC filter configuration callback
- ***void(* __FMAC_HandleTypeDef::FilterPreloadCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC filter preload callback
- ***void(* __FMAC_HandleTypeDef::MspInitCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC Msp Init callback
- ***void(* __FMAC_HandleTypeDef::MspDeInitCallback)(struct __FMAC_HandleTypeDef *hfmac)***
  FMAC Msp DeInit callback
- ***HAL_LockTypeDef __FMAC_HandleTypeDef::Lock***
  FMAC locking object
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::State***
  FMAC state related to global handle management This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::RdState***
  FMAC state related to read operations (access to Y buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO HAL_FMAC_StateTypeDef __FMAC_HandleTypeDef::WrState***
  FMAC state related to write operations (access to X1 buffer) This parameter can be a value of **HAL_FMAC_StateTypeDef**
- ***__IO uint32_t __FMAC_HandleTypeDef::ErrorCode***
  FMAC peripheral error code This parameter can be a value of ***FMAC_Error_Code***

28.1.2 FMAC_FilterConfigTypeDef

FMAC_FilterConfigTypeDef is defined in the stm32u5xx_hal_fmac.h

Data Fields

- ***uint8_t InputBaseAddress***
- ***uint8_t InputBufferSize***
- ***uint32_t InputThreshold***

- ***uint8_t CoeffBaseAddress***
- ***uint8_t CoeffBufferSize***
- ***uint8_t OutputBaseAddress***
- ***uint8_t OutputBufferSize***
- ***uint32_t OutputThreshold***
- ***int16_t * pCoeffA***
- ***uint8_t CoeffASize***
- ***int16_t * pCoeffB***
- ***uint8_t CoeffBSize***
- ***uint8_t InputAccess***
- ***uint8_t OutputAccess***
- ***uint32_t Clip***
- ***uint32_t Filter***
- ***uint8_t P***
- ***uint8_t Q***
- ***uint8_t R***

Field Documentation

- ***uint8_t FMAC_FilterConfigTypeDef::InputBaseAddress***
  Base address of the input buffer (X1) within the internal memory (0x00 to 0xFF). Ignored if InputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::InputBufferSize***
  Number of 16-bit words allocated to the input buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::InputThreshold***
  Input threshold: the buffer full flag will be set if the number of free spaces in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBaseAddress***
  Base address of the coefficient buffer (X2) within the internal memory (0x00 to 0xFF). Ignored if CoeffBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBufferSize***
  Number of 16-bit words allocated to the coefficient buffer. 0 if a previous configuration should be kept.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBaseAddress***
  Base address of the output buffer (Y) within the internal memory (0x00 to 0xFF). Ignored if OuputBufferSize is set to 0 (previous configuration kept). Note: the buffers can overlap or even coincide exactly.
- ***uint8_t FMAC_FilterConfigTypeDef::OutputBufferSize***
  Number of 16-bit words allocated to the output buffer (including the optional "headroom"). 0 if a previous configuration should be kept.
- ***uint32_t FMAC_FilterConfigTypeDef::OutputThreshold***
  Output threshold: the buffer empty flag will be set if the number of unread values in the buffer is lower than this threshold. This parameter can be a value of ***FMAC_Data_Buffer_Threshold***.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffA***
  [IIR only] Initialization of the coefficient vector A. If not needed, it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffASize***
  Size of the coefficient vector A.
- ***int16_t* FMAC_FilterConfigTypeDef::pCoeffB***
  Initialization of the coefficient vector B. If not needed (re-use of a previously loaded buffer), it should be set to NULL.
- ***uint8_t FMAC_FilterConfigTypeDef::CoeffBSize***
  Size of the coefficient vector B.
- ***uint8_t FMAC_FilterConfigTypeDef::InputAccess***
  Access to the input buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.

- ***uint8_t FMAC_FilterConfigTypeDef::OutputAccess***
  Access to the output buffer (internal memory area): DMA, IT, Polling, None. This parameter can be a value of ***FMAC_Buffer_Access***.
- ***uint32_t FMAC_FilterConfigTypeDef::Clip***
  Enable or disable the clipping feature. If the q1.15 range is exceeded, wrapping is done when the clipping feature is disabled and saturation is done when the clipping feature is enabled. This parameter can be a value of ***FMAC_Clip_State***.
- ***uint32_t FMAC_FilterConfigTypeDef::Filter***
  Filter type. This parameter can be a value of ***FMAC_Functions*** (filter related values).
- ***uint8_t FMAC_FilterConfigTypeDef::P***
  Parameter P (vector length, number of filter taps, etc.).
- ***uint8_t FMAC_FilterConfigTypeDef::Q***
  Parameter Q (vector length, etc.). Ignored if not needed.
- ***uint8_t FMAC_FilterConfigTypeDef::R***
  Parameter R (gain, etc.). Ignored if not needed.

28.2 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

28.2.1 How to use this driver

The FMAC HAL driver can be used as follows:

1. Initialize the FMAC low level resources by implementing the HAL_FMAC_MspInit():
   - Enable the FMAC interface clock using __HAL_RCC_FMAC_CLK_ENABLE().
   - In case of using interrupts (e.g. access configured as FMAC_BUFFER_ACCESS_IT):
     - Configure the FMAC interrupt priority using HAL_NVIC_SetPriority().
     - Enable the FMAC IRQ handler using HAL_NVIC_EnableIRQ().
     - In FMAC IRQ handler, call HAL_FMAC_IRQHandler().
   - In case of using DMA to control data transfer (e.g. access configured as FMAC_BUFFER_ACCESS_DMA):
     - Enable the DMA interface clock using __HAL_RCC_DMA1_CLK_ENABLE() or __HAL_RCC_DMA2_CLK_ENABLE() depending on the used DMA instance.
     - Enable the DMAMUX1 interface clock using __HAL_RCC_DMAMUX1_CLK_ENABLE().
     - If the initialization of the internal buffers (coefficients, input, output) is done via DMA, configure and enable one DMA channel for managing data transfer from memory to memory (preload channel).
     - If the input buffer is accessed via DMA, configure and enable one DMA channel for managing data transfer from memory to peripheral (input channel).
     - If the output buffer is accessed via DMA, configure and enable one DMA channel for managing data transfer from peripheral to memory (output channel).
     - Associate the initialized DMA handle(s) to the FMAC DMA handle(s) using __HAL_LINKDMA().
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the enabled DMA channel(s) using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Initialize the FMAC HAL using HAL_FMAC_Init(). This function resorts to HAL_FMAC_MspInit() for low-level initialization.

3. Configure the FMAC processing (filter) using HAL_FMAC_FilterConfig() or HAL_FMAC_FilterConfig_DMA(). This function:
   – Defines the memory area within the FMAC internal memory (input, coefficients, output) and the associated threshold (input, output).
   – Configures the filter and its parameters:
     ◦ Finite Impulse Response (FIR) filter (also known as convolution).
     ◦ Infinite Impulse Response (IIR) filter (direct form 1).
   – Choose the way to access to the input and output buffers: none, polling, DMA, IT. "none" means the input and/or output data will be handled by another IP (ADC, DAC, etc.).
   – Enable the error interruptions in the input access and/or the output access is done through IT/DMA. If an error occurs, the interruption will be triggered in loop. In order to recover, the user will have to reset the IP with the sequence HAL_FMAC_DeInit / HAL_FMAC_Init. Optionally, he can also disable the interrupt using __HAL_FMAC_DISABLE_IT; the error status will be kept, but no more interrupt will be triggered.
   – Write the provided coefficients into the internal memory using polling mode (HAL_FMAC_FilterConfig()) or DMA (HAL_FMAC_FilterConfig_DMA()). In the DMA case, HAL_FMAC_FilterConfigCallback() is called when the handling is over.
4. Optionally, the user can enable the error interruption related to saturation by calling __HAL_FMAC_ENABLE_IT. This helps in debugging the filter. If a saturation occurs, the interruption will be triggered in loop. In order to recover, the user will have to:
   – Disable the interruption by calling __HAL_FMAC_DISABLE_IT if the user wishes to continue all the same.
   – Reset the IP with the sequence HAL_FMAC_DeInit / HAL_FMAC_Init.
5. Optionally, preload input (FIR, IIR) and output (IIR) data using HAL_FMAC_FilterPreload() or HAL_FMAC_FilterPreload_DMA(). In the DMA case, HAL_FMAC_FilterPreloadCallback() is called when the handling is over. This step is optional as the filter can be started without preloaded data.
6. Start the FMAC processing (filter) using HAL_FMAC_FilterStart(). This function also configures the output buffer that will be filled from the circular internal output buffer. The function returns immediately without updating the provided buffer. The IP processing will be active until HAL_FMAC_FilterStop() is called.
7. If the input internal buffer is accessed via DMA, HAL_FMAC_HalfGetDataCallback() will be called to indicate that half of the input buffer has been handled.
8. If the input internal buffer is accessed via DMA or interrupt, HAL_FMAC_GetDataCallback() will be called to require new input data. It will be provided through HAL_FMAC_AppendFilterData() if the DMA isn't in circular mode.
9. If the output internal buffer is accessed via DMA, HAL_FMAC_HalfOutputDataReadyCallback() will be called to indicate that half of the output buffer has been handled.
10. If the output internal buffer is accessed via DMA or interrupt, HAL_FMAC_OutputDataReadyCallback() will be called to require a new output buffer. It will be provided through HAL_FMAC_ConfigFilterOutputBuffer() if the DMA isn't in circular mode.
11. In all modes except none, provide new input data to be processed via HAL_FMAC_AppendFilterData(). This function should only be called once the previous input data has been handled (the preloaded input data isn't concerned).
12. In all modes except none, provide a new output buffer to be filled via HAL_FMAC_ConfigFilterOutputBuffer(). This function should only be called once the previous user's output buffer has been filled.
13. In polling mode, handle the input and output data using HAL_FMAC_PollFilterData(). This function:
    – Write the user's input data (provided via HAL_FMAC_AppendFilterData()) into the FMAC input memory area.
    – Read the FMAC output memory area and write it into the user's output buffer. It will return either when:
    – the user's output buffer is filled.
    – the user's input buffer has been handled. The unused data (unread input data or free output data) will not be saved. The user will have to use the updated input and output sizes to keep track of them.
14. Stop the FMAC processing (filter) using HAL_FMAC_FilterStop().
15. Call HAL_FMAC_DeInit() to de-initialize the FMAC peripheral. This function resorts to HAL_FMAC_MspDeInit() for low-level de-initialization.

28.2.2 Callback registration

The compilation define USE_HAL_FMAC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_FMAC_RegisterCallback() to register a user callback. Function HAL_FMAC_RegisterCallback() allows to register following callbacks:

- ErrorCallback : Error Callback.
- HalfGetDataCallback : Get Half Data Callback.
- GetDataCallback : Get Data Callback.
- HalfOutputDataReadyCallback : Half Output Data Ready Callback.
- OutputDataReadyCallback : Output Data Ready Callback.
- FilterConfigCallback : Filter Configuration Callback.
- FilterPreloadCallback : Filter Preload Callback.
- MspInitCallback : FMAC MspInit.
- MspDeInitCallback : FMAC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_FMAC_UnRegisterCallback() to reset a callback to the default weak function. HAL_FMAC_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : Error Callback.
- HalfGetDataCallback : Get Half Data Callback.
- GetDataCallback : Get Data Callback.
- HalfOutputDataReadyCallback : Half Output Data Ready Callback.
- OutputDataReadyCallback : Output Data Ready Callback.
- FilterConfigCallback : Filter Configuration Callback.
- FilterPreloadCallback : Filter Preload Callback.
- MspInitCallback : FMAC MspInit.
- MspDeInitCallback : FMAC MspDeInit.

By default, after the HAL_FMAC_Init() and when the state is HAL_FMAC_STATE_RESET all callbacks are set to the corresponding weak functions: examples GetDataCallback(), OutputDataReadyCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_FMAC_Init() and HAL_FMAC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_FMAC_Init() and HAL_FMAC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_FMAC_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_FMAC_STATE_READY or HAL_FMAC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_FMAC_RegisterCallback() before calling HAL_FMAC_DeInit() or HAL_FMAC_Init() function.

When the compilation define USE_HAL_FMAC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

28.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the FMAC peripheral and the associated handle
- DeInitialize the FMAC peripheral
- Initialize the FMAC MSP (MCU Specific Package)
- De-Initialize the FMAC MSP
- Register a User FMAC Callback
- Unregister a FMAC CallBack

This section contains the following APIs:

- ***HAL_FMAC_Init()***
- ***HAL_FMAC_DeInit()***
- ***HAL_FMAC_MspInit()***

- *HAL_FMAC_MspDeInit()*
- *HAL_FMAC_RegisterCallback()*
- *HAL_FMAC_UnRegisterCallback()*

28.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the FMAC peripheral: memory area, filter type and parameters, way to access to the input and output memory area (none, polling, IT, DMA).
- Start the FMAC processing (filter).
- Handle the input data that will be provided into FMAC.
- Handle the output data provided by FMAC.
- Stop the FMAC processing (filter).

This section contains the following APIs:

- *HAL_FMAC_FilterConfig()*
- *HAL_FMAC_FilterConfig_DMA()*
- *HAL_FMAC_FilterPreload()*
- *HAL_FMAC_FilterPreload_DMA()*
- *HAL_FMAC_FilterStart()*
- *HAL_FMAC_AppendFilterData()*
- *HAL_FMAC_ConfigFilterOutputBuffer()*
- *HAL_FMAC_PollFilterData()*
- *HAL_FMAC_FilterStop()*

28.2.5 Callback functions

This section provides Interruption and DMA callback functions:

- DMA or Interrupt: the user's input data is half written (DMA only) or completely written.
- DMA or Interrupt: the user's output buffer is half filled (DMA only) or completely filled.
- DMA or Interrupt: error handling.

This section contains the following APIs:

- *HAL_FMAC_ErrorCallback()*
- *HAL_FMAC_HalfGetDataCallback()*
- *HAL_FMAC_GetDataCallback()*
- *HAL_FMAC_HalfOutputDataReadyCallback()*
- *HAL_FMAC_OutputDataReadyCallback()*
- *HAL_FMAC_FilterConfigCallback()*
- *HAL_FMAC_FilterPreloadCallback()*

28.2.6 IRQ handler management

This section provides IRQ handler function.

This section contains the following APIs:

- *HAL_FMAC_IRQHandler()*

28.2.7 Peripheral State and Error functions

This subsection provides functions allowing to

- Check the FMAC state
- Get error code

This section contains the following APIs:

- *HAL_FMAC_GetState()*
- *HAL_FMAC_GetError()*

28.2.8 Detailed description of functions

HAL_FMAC_Init

Function name

HAL_StatusTypeDef HAL_FMAC_Init (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC peripheral and the associated handle.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_DeInit

Function name

HAL_StatusTypeDef HAL_FMAC_DeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC peripheral.

Parameters

- **hfmac:** pointer to a FMAC structure.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_MspInit

Function name

void HAL_FMAC_MspInit (FMAC_HandleTypeDef * hfmac)

Function description

Initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_MspDeInit

Function name

void HAL_FMAC_MspDeInit (FMAC_HandleTypeDef * hfmac)

Function description

De-initialize the FMAC MSP.

Parameters

- **hfmac:** FMAC handle.

Return values

- **None:**

HAL_FMAC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_FMAC_RegisterCallback (FMAC_HandleTypeDef * hfmac, HAL_FMAC_CallbackIDTypeDef CallbackID, pFMAC_CallbackTypeDef pCallback)

Function description

Register a User FMAC Callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_FMAC_ERROR_CB_ID Error Callback ID
  - HAL_FMAC_HALF_GET_DATA_CB_ID Get Half Data Callback ID
  - HAL_FMAC_GET_DATA_CB_ID Get Data Callback ID
  - HAL_FMAC_HALF_OUTPUT_DATA_READY_CB_ID Half Output Data Ready Callback ID
  - HAL_FMAC_OUTPUT_DATA_READY_CB_ID Output Data Ready Callback ID
  - HAL_FMAC_FILTER_CONFIG_CB_ID Filter Configuration Callback ID
  - HAL_FMAC_FILTER_PRELOAD_CB_ID Filter Preload Callback ID
  - HAL_FMAC_MSPINIT_CB_ID FMAC MspInit ID
  - HAL_FMAC_MSPDEINIT_CB_ID FMAC MspDeInit ID
- **pCallback:** pointer to the Callback function.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The User FMAC Callback is to be used instead of the weak predefined callback.
- The HAL_FMAC_RegisterCallback() may be called before HAL_FMAC_Init() in HAL_FMAC_STATE_RESET to register callbacks for HAL_FMAC_MSPINIT_CB_ID and HAL_FMAC_MSPDEINIT_CB_ID.

HAL_FMAC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_FMAC_UnRegisterCallback (FMAC_HandleTypeDef * hfmac, HAL_FMAC_CallbackIDTypeDef CallbackID)

Function description

Unregister a FMAC CallBack.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_FMAC_ERROR_CB_ID Error Callback ID
  - HAL_FMAC_HALF_GET_DATA_CB_ID Get Half Data Callback ID
  - HAL_FMAC_GET_DATA_CB_ID Get Data Callback ID
  - HAL_FMAC_HALF_OUTPUT_DATA_READY_CB_ID Half Output Data Ready Callback ID
  - HAL_FMAC_OUTPUT_DATA_READY_CB_ID Output Data Ready Callback ID
  - HAL_FMAC_FILTER_CONFIG_CB_ID Filter Configuration Callback ID
  - HAL_FMAC_FILTER_PRELOAD_CB_ID Filter Preload Callback ID
  - HAL_FMAC_MSPINIT_CB_ID FMAC MspInit ID
  - HAL_FMAC_MSPDEINIT_CB_ID FMAC MspDeInit ID

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The FMAC callback is redirected to the weak predefined callback.
- The HAL_FMAC_UnRegisterCallback() may be called before HAL_FMAC_Init() in HAL_FMAC_STATE_RESET to register callbacks for HAL_FMAC_MSPINIT_CB_ID and HAL_FMAC_MSPDEINIT_CB_ID.

HAL_FMAC_FilterConfig

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using polling mode.

HAL_FMAC_FilterConfig_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterConfig_DMA (FMAC_HandleTypeDef * hfmac, FMAC_FilterConfigTypeDef * pConfig)

Function description

Configure the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pConfig:** pointer to a FMAC_FilterConfigTypeDef structure that contains the FMAC configuration information.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The configuration is done according to the parameters specified in the FMAC_FilterConfigTypeDef structure. The provided data will be loaded using DMA.

HAL_FMAC_FilterPreload

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using polling mode.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterPreload_DMA

Function name

HAL_StatusTypeDef HAL_FMAC_FilterPreload_DMA (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint8_t InputSize, int16_t * pOutput, uint8_t OutputSize)

Function description

Preload the input (FIR, IIR) and output data (IIR) of the FMAC filter.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** Preloading of the first elements of the input buffer (X1). If not needed (no data available when starting), it should be set to NULL.
- **InputSize:** Size of the input vector. As pInput is used for preloading data, it cannot be bigger than the input memory area.
- **pOutput:** [IIR] Preloading of the first elements of the output vector (Y). If not needed, it should be set to NULL.
- **OutputSize:** Size of the output vector. As pOutput is used for preloading data, it cannot be bigger than the output memory area.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- The set(s) of data will be used by FMAC as soon as HAL_FMAC_FilterStart is called. The provided data will be loaded using DMA.
- The input and the output buffers can be filled by calling several times HAL_FMAC_FilterPreload (each call filling partly the buffers). In case of overflow (too much data provided through all these calls), an error will be returned.

HAL_FMAC_FilterStart

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStart (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Start the FMAC processing according to the existing FMAC configuration.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** pointer to buffer where output data of FMAC processing will be stored in the next steps. If it is set to NULL, the output will not be read and it will be up to an external IP to empty the output buffer.
- **pOutputSize:** pointer to the size of the output buffer. The number of read data will be written here.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_AppendFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_AppendFilterData (FMAC_HandleTypeDef * hfmac, int16_t * pInput, uint16_t * pInputSize)

Function description

Provide a new input buffer that will be loaded into the FMAC input memory area.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pInput:** New input vector (additional input data).
- **pInputSize:** Size of the input vector (if all the data can't be written, it will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ConfigFilterOutputBuffer

Function name

HAL_StatusTypeDef HAL_FMAC_ConfigFilterOutputBuffer (FMAC_HandleTypeDef * hfmac, int16_t * pOutput, uint16_t * pOutputSize)

Function description

Provide a new output buffer to be filled with the data computed by FMAC unit.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **pOutput:** New output vector.
- **pOutputSize:** Size of the output vector (if the vector can't be entirely filled, pOutputSize will be updated with the number of data read from FMAC).

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_PollFilterData

Function name

HAL_StatusTypeDef HAL_FMAC_PollFilterData (FMAC_HandleTypeDef * hfmac, uint32_t Timeout)

Function description

Handle the input and/or output data in polling mode.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.
- **Timeout:** timeout value.

Return values

- **HAL_StatusTypeDef:** HAL status

Notes

- This function writes the previously provided user's input data and fills the previously provided user's output buffer, according to the existing FMAC configuration (polling mode only). The function returns when the input data has been handled or when the output data is filled. The possible unused data isn't kept. It will be up to the user to handle it. The previously provided pInputSize and pOutputSize will be used to indicate to the size of the read/written data to the user.

HAL_FMAC_FilterStop

Function name

HAL_StatusTypeDef HAL_FMAC_FilterStop (FMAC_HandleTypeDef * hfmac)

Function description

Stop the FMAC processing.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_StatusTypeDef:** HAL status

HAL_FMAC_ErrorCallback

Function name

void HAL_FMAC_ErrorCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC error callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfGetDataCallback

Function name

void HAL_FMAC_HalfGetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get half data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetDataCallback

Function name

void HAL_FMAC_GetDataCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC get data callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_HalfOutputDataReadyCallback

Function name

void HAL_FMAC_HalfOutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC half output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_OutputDataReadyCallback

Function name

void HAL_FMAC_OutputDataReadyCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC output data ready callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterConfigCallback

Function name

void HAL_FMAC_FilterConfigCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter configuration callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_FilterPreloadCallback

Function name

void HAL_FMAC_FilterPreloadCallback (FMAC_HandleTypeDef * hfmac)

Function description

FMAC filter preload callback.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_IRQHandler

Function name

void HAL_FMAC_IRQHandler (FMAC_HandleTypeDef * hfmac)

Function description

Handle FMAC interrupt request.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **None:**

HAL_FMAC_GetState

Function name

HAL_FMAC_StateTypeDef HAL_FMAC_GetState (const FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC state.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **HAL_FMAC_StateTypeDef:** FMAC state

HAL_FMAC_GetError

Function name

uint32_t HAL_FMAC_GetError (const FMAC_HandleTypeDef * hfmac)

Function description

Return the FMAC peripheral error.

Parameters

- **hfmac:** pointer to a FMAC_HandleTypeDef structure that contains the configuration information for FMAC module.

Return values

- **uint32_t:** Error bit-map based on FMAC Error code

Notes

- The returned error is a bit-map combination of possible errors.

28.3 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

28.3.1 FMAC

FMAC

FMAC Buffer Access

FMAC_BUFFER_ACCESS_NONE

Buffer handled by an external IP (ADC for instance)

FMAC_BUFFER_ACCESS_DMA

Buffer accessed through DMA

FMAC_BUFFER_ACCESS_POLLING

Buffer accessed through polling

FMAC_BUFFER_ACCESS_IT

Buffer accessed through interruptions

FMAC Clip State

FMAC_CLIP_DISABLED

Clipping disabled

FMAC_CLIP_ENABLED

Clipping enabled

FMAC Data Buffer Threshold

FMAC_THRESHOLD_1

Input: Buffer full flag set if the number of free spaces in the buffer is less than 1. Output: Buffer empty flag set if the number of unread values in the buffer is less than 1.

FMAC_THRESHOLD_2

Input: Buffer full flag set if the number of free spaces in the buffer is less than 2. Output: Buffer empty flag set if the number of unread values in the buffer is less than 2.

FMAC_THRESHOLD_4

Input: Buffer full flag set if the number of free spaces in the buffer is less than 4. Output: Buffer empty flag set if the number of unread values in the buffer is less than 4.

FMAC_THRESHOLD_8

Input: Buffer full flag set if the number of free spaces in the buffer is less than 8. Output: Buffer empty flag set if the number of unread values in the buffer is less than 8.

FMAC_THRESHOLD_NO_VALUE

The configured threshold value shouldn't be changed

FMAC Error code

HAL_FMAC_ERROR_NONE

No error

HAL_FMAC_ERROR_SAT

Saturation error

HAL_FMAC_ERROR_UNFL

Underflow error

HAL_FMAC_ERROR_OVFL

Overflow error

HAL_FMAC_ERROR_DMA

DMA error

HAL_FMAC_ERROR_RESET

Reset error

HAL_FMAC_ERROR_PARAM

Parameter error

HAL_FMAC_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_FMAC_ERROR_TIMEOUT

Timeout error

FMAC Exported Macros

__HAL_FMAC_RESET_HANDLE_STATE

Description:
- Reset FMAC handle state.

Parameters:
- __HANDLE__: FMAC handle.

Return value:
- None

__HAL_FMAC_ENABLE_IT

Description:
- Enable the specified FMAC interrupt.

Parameters:
- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:
- None

__HAL_FMAC_DISABLE_IT

Description:
- Disable the FMAC interrupt.

Parameters:
- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC Interrupt. This parameter can be any combination of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:
- None

__HAL_FMAC_GET_IT

Description:

- Check whether the specified FMAC interrupt occurred or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be any combination of the following values:
  – FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  – FMAC_FLAG_X1FULL X1 Buffer Full Flag
  – FMAC_FLAG_OVFL Overflow Error Flag
  – FMAC_FLAG_UNFL Underflow Error Flag
  – FMAC_FLAG_SAT Saturation Error Flag

Return value:

- SET: (interrupt occurred) or RESET (interrupt did not occurred)

__HAL_FMAC_CLEAR_IT

Description:

- Clear specified FMAC interrupt status.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to clear.

Return value:

- None

__HAL_FMAC_GET_FLAG

Description:

- Check whether the specified FMAC status flag is set or not.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to check. This parameter can be any combination of the following values:
  – FMAC_FLAG_YEMPTY Y Buffer Empty Flag
  – FMAC_FLAG_X1FULL X1 Buffer Full Flag
  – FMAC_FLAG_OVFL Overflow Error Flag
  – FMAC_FLAG_UNFL Underflow Error Flag
  – FMAC_FLAG_SAT Saturation error Flag

Return value:

- SET: (flag is set) or RESET (flag is reset)

__HAL_FMAC_CLEAR_FLAG

Description:

- Clear specified FMAC status flag.

Parameters:

- __HANDLE__: FMAC handle.
- __FLAG__: FMAC flag to clear.

Return value:

- None

__HAL_FMAC_GET_IT_SOURCE

Description:

- Check whether the specified FMAC interrupt is enabled or not.

Parameters:

- __HANDLE__: FMAC handle.
- __INTERRUPT__: FMAC interrupt to check. This parameter can be one of the following values:
  - FMAC_IT_RIEN Read interrupt enable
  - FMAC_IT_WIEN Write interrupt enable
  - FMAC_IT_OVFLIEN Overflow error interrupt enable
  - FMAC_IT_UNFLIEN Underflow error interrupt enable
  - FMAC_IT_SATIEN Saturation error interrupt enable (this helps in debugging a filter)

Return value:

- FlagStatus

FMAC status flags

FMAC_FLAG_YEMPTY

Y Buffer Empty Flag

FMAC_FLAG_X1FULL

X1 Buffer Full Flag

FMAC_FLAG_OVFL

Overflow Error Flag

FMAC_FLAG_UNFL

Underflow Error Flag

FMAC_FLAG_SAT

Saturation Error Flag (this helps in debugging a filter)

FMAC Functions

FMAC_FUNC_LOAD_X1

Load X1 buffer

FMAC_FUNC_LOAD_X2

Load X2 buffer

FMAC_FUNC_LOAD_Y

Load Y buffer

FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

FMAC Interrupts Enable bit

FMAC_IT_RIEN

Read Interrupt Enable

FMAC_IT_WIEN

Write Interrupt Enable

FMAC_IT_OVFLIEN

Overflow Error Interrupt Enable

FMAC_IT_UNFLIEN

Underflow Error Interrupt Enable

FMAC_IT_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

29 HAL GFXMMU Generic Driver

29.1 GFXMMU Firmware driver registers structures

29.1.1 GFXMMU_BuffersTypeDef

GFXMMU_BuffersTypeDef is defined in the stm32u5xx_hal_gfxmmu.h

Data Fields

- ***uint32_t Buf0Address***
- ***uint32_t Buf1Address***
- ***uint32_t Buf2Address***
- ***uint32_t Buf3Address***

Field Documentation

- ***uint32_t GFXMMU_BuffersTypeDef::Buf0Address***
  Physical address of buffer 0.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf1Address***
  Physical address of buffer 1.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf2Address***
  Physical address of buffer 2.
- ***uint32_t GFXMMU_BuffersTypeDef::Buf3Address***
  Physical address of buffer 3.

29.1.2 GFXMMU_AddressCacheTypeDef

GFXMMU_AddressCacheTypeDef is defined in the stm32u5xx_hal_gfxmmu.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t AddressCacheLockBuffer***

Field Documentation

- ***FunctionalState GFXMMU_AddressCacheTypeDef::Activation***
  Address Cache and enable/disable.
  Note:
  – : All following parameters are useful only if address cache is enabled.
- ***uint32_t GFXMMU_AddressCacheTypeDef::AddressCacheLockBuffer***
  Buffer on which the address cache is locked. This parameter can be a value of ***GFXMMU_AddressCacheLockBuffer***.
  Note:
  – : Useful only when lock of the address cache is enabled.

29.1.3 GFXMMU_InterruptsTypeDef

GFXMMU_InterruptsTypeDef is defined in the stm32u5xx_hal_gfxmmu.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t UsedInterrupts***

Field Documentation

- ***FunctionalState GFXMMU_InterruptsTypeDef::Activation***
  Interrupts enable/disable.
- ***uint32_t GFXMMU_InterruptsTypeDef::UsedInterrupts***
  Interrupts used. This parameter can be a values combination of ***GFXMMU_Interrupts***.
  Note:
  – : Useful only when interrupts are enabled.

29.1.4 GFXMMU_InitTypeDef

GFXMMU_InitTypeDef is defined in the stm32u5xx_hal_gfxmmu.h

Data Fields

- ***uint32_t BlocksPerLine***
- ***uint32_t DefaultValue***
- ***GFXMMU_BuffersTypeDef Buffers***
- ***GFXMMU_AddressCacheTypeDef AddressCache***
- ***GFXMMU_InterruptsTypeDef Interrupts***

Field Documentation

- ***uint32_t GFXMMU_InitTypeDef::BlocksPerLine***
  Number of blocks of 16 bytes per line. This parameter can be a value of ***GFXMMU_BlocksPerLine***.
- ***uint32_t GFXMMU_InitTypeDef::DefaultValue***
  Value returned when virtual memory location not physically mapped.
- ***GFXMMU_BuffersTypeDef GFXMMU_InitTypeDef::Buffers***
  Physical buffers addresses.
- ***GFXMMU_AddressCacheTypeDef GFXMMU_InitTypeDef::AddressCache***
  Address Cache parameters.
- ***GFXMMU_InterruptsTypeDef GFXMMU_InitTypeDef::Interrupts***
  Interrupts parameters.

29.1.5 __GFXMMU_HandleTypeDef

__GFXMMU_HandleTypeDef is defined in the stm32u5xx_hal_gfxmmu.h

Data Fields

- ***GFXMMU_TypeDef * Instance***
- ***GFXMMU_InitTypeDef Init***
- ***HAL_GFXMMU_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***GFXMMU_TypeDef* __GFXMMU_HandleTypeDef::Instance***
  GFXMMU instance.
- ***GFXMMU_InitTypeDef __GFXMMU_HandleTypeDef::Init***
  GFXMMU init parameters.
- ***HAL_GFXMMU_StateTypeDef __GFXMMU_HandleTypeDef::State***
  GFXMMU state.
- ***__IO uint32_t __GFXMMU_HandleTypeDef::ErrorCode***
  GFXMMU error code.
- ***void(* __GFXMMU_HandleTypeDef::ErrorCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU error callback.
- ***void(* __GFXMMU_HandleTypeDef::MspInitCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU MSP init callback.
- ***void(* __GFXMMU_HandleTypeDef::MspDeInitCallback)(struct __GFXMMU_HandleTypeDef *hgfxmmu)***
  GFXMMU MSP de-init callback.

29.1.6 GFXMMU_LutLineTypeDef

GFXMMU_LutLineTypeDef is defined in the stm32u5xx_hal_gfxmmu.h

Data Fields

- ***uint32_t LineNumber***
- ***uint32_t LineStatus***

- ***uint32_t FirstVisibleBlock***
- ***uint32_t LastVisibleBlock***
- ***int32_t LineOffset***

Field Documentation

- ***uint32_t GFXMMU_LutLineTypeDef::LineNumber***
  LUT line number. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- ***uint32_t GFXMMU_LutLineTypeDef::LineStatus***
  LUT line enable/disable. This parameter can be a value of ***GFXMMU_LutLineStatus***.
- ***uint32_t GFXMMU_LutLineTypeDef::FirstVisibleBlock***
  First visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***uint32_t GFXMMU_LutLineTypeDef::LastVisibleBlock***
  Last visible block on this line. This parameter must be a number between Min_Data = 0 and Max_Data = 255.
- ***int32_t GFXMMU_LutLineTypeDef::LineOffset***
  Offset of block 0 of the current line in physical buffer. This parameter must be a number between Min_Data = -4080 and Max_Data = 4190208.
  Note:
  – : Line offset has to be computed with the following formula: LineOffset = [(Blocks already used) - (1st visible block)]*BlockSize.

29.2 GFXMMU Firmware driver API description

The following section lists the various functions of the GFXMMU library.

29.2.1 How to use this driver

Initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspInit() :
   – Enable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   – If interrupts are used, enable and configure GFXMMU global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
2. Configure the number of blocks per line, default value, physical buffer addresses, cache and pre-fetch parameters and interrupts using the HAL_GFXMMU_Init() function.

LUT configuration

1. Use HAL_GFXMMU_DisableLutLines() to deactivate all LUT lines (or a range of lines).
2. Use HAL_GFXMMU_ConfigLut() to copy LUT from flash to look up RAM.
3. Use HAL_GFXMMU_ConfigLutLine() to configure one line of LUT.

Force flush and/or invalidate of cache

1. Use HAL_GFXMMU_ConfigForceCache() to flush and/or invalidate cache.

Modify physical buffer addresses

1. Use HAL_GFXMMU_ModifyBuffers() to modify physical buffer addresses.

Modify cache and pre-fetch parameters

1. Use HAL_GFXMMU_ModifyCachePrefetch() to modify cache and pre-fetch parameters.

Modify address cache parameters

NOTE : This feature is only available on STM32U5F9/STM32U5G9 devices.

1. Use HAL_GFXMMU_ModifyAddressCache() to modify address cache parameters.

Error management

1. If interrupts are used, HAL_GFXMMU_IRQHandler() will be called when an error occurs. This function will call HAL_GFXMMU_ErrorCallback(). Use HAL_GFXMMU_GetError() to get the error code.

De-initialization

1. As prerequisite, fill in the HAL_GFXMMU_MspDeInit() :
   – Disable GFXMMU clock interface with __HAL_RCC_GFXMMU_CLK_ENABLE().
   – If interrupts has been used, disable GFXMMU global interrupt with HAL_NVIC_DisableIRQ().
2. De-initialize GFXMMU using the HAL_GFXMMU_DeInit() function.

Callback registration

The compilation define USE_HAL_GFXMMU_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_GFXMMU_RegisterCallback() to register a user callback.

Function HAL_GFXMMU_RegisterCallback() allows to register following callbacks:

- ErrorCallback : GFXMMU error.
- MspInitCallback : GFXMMU MspInit.
- MspDeInitCallback : GFXMMU MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_GFXMMU_UnRegisterCallback() to reset a callback to the default weak (overridden) function. HAL_GFXMMU_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- ErrorCallback : GFXMMU error.
- MspInitCallback : GFXMMU MspInit.
- MspDeInitCallback : GFXMMU MspDeInit.

By default, after the HAL_GFXMMU_Init and if the state is HAL_GFXMMU_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions: examples HAL_GFXMMU_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_GFXMMU_Init and HAL_GFXMMU_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_GFXMMU_Init and HAL_GFXMMU_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_GFXMMU_RegisterCallback before calling HAL_GFXMMU_DeInit or HAL_GFXMMU_Init function.

When the compilation define USE_HAL_GFXMMU_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

29.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the GFXMMU.
- De-initialize the GFXMMU.

This section contains the following APIs:

- *HAL_GFXMMU_Init()*
- *HAL_GFXMMU_DeInit()*
- *HAL_GFXMMU_MspInit()*
- *HAL_GFXMMU_MspDeInit()*
- *HAL_GFXMMU_RegisterCallback()*
- *HAL_GFXMMU_UnRegisterCallback()*

29.2.3 Operation functions

This section provides functions allowing to:
- Configure LUT.
- Force flush and/or invalidate of cache.
- Modify physical buffer addresses.
- Modify cache and pre-fetch parameters.
- Manage error.

This section contains the following APIs:
- *HAL_GFXMMU_ConfigLut()*
- *HAL_GFXMMU_DisableLutLines()*
- *HAL_GFXMMU_ConfigLutLine()*
- *HAL_GFXMMU_ModifyBuffers()*
- *HAL_GFXMMU_ModifyAddressCache()*
- *HAL_GFXMMU_IRQHandler()*
- *HAL_GFXMMU_ErrorCallback()*

29.2.4 State functions

This section provides functions allowing to:
- Get GFXMMU handle state.
- Get GFXMMU error code.

This section contains the following APIs:
- *HAL_GFXMMU_GetState()*
- *HAL_GFXMMU_GetError()*

29.2.5 Detailed description of functions

HAL_GFXMMU_Init

Function name

HAL_StatusTypeDef HAL_GFXMMU_Init (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

Initialize the GFXMMU according to the specified parameters in the GFXMMU_InitTypeDef structure and initialize the associated handle.

Parameters
- **hgfxmmu:** GFXMMU handle.

Return values
- **HAL:** status.

HAL_GFXMMU_DeInit

Function name

HAL_StatusTypeDef HAL_GFXMMU_DeInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

De-initialize the GFXMMU.

Parameters
- **hgfxmmu:** GFXMMU handle.

Return values
- **HAL:** status.

HAL_GFXMMU_MspInit

Function name

void HAL_GFXMMU_MspInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

Initialize the GFXMMU MSP.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_MspDeInit

Function name

void HAL_GFXMMU_MspDeInit (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

De-initialize the GFXMMU MSP.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_RegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXMMU_RegisterCallback (GFXMMU_HandleTypeDef * hgfxmmu, HAL_GFXMMU_CallbackIDTypeDef CallbackID, pGFXMMU_CallbackTypeDef pCallback)

Function description

Register a user GFXMMU callback to be used instead of the weak predefined callback.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_GFXMMU_ERROR_CB_ID error callback ID.
  - HAL_GFXMMU_MSPINIT_CB_ID MSP init callback ID.
  - HAL_GFXMMU_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_GFXMMU_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXMMU_UnRegisterCallback (GFXMMU_HandleTypeDef * hgfxmmu, HAL_GFXMMU_CallbackIDTypeDef CallbackID)

Function description

Unregister a user GFXMMU callback.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_GFXMMU_ERROR_CB_ID error callback ID.
  - HAL_GFXMMU_MSPINIT_CB_ID MSP init callback ID.
  - HAL_GFXMMU_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_GFXMMU_ConfigLut

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigLut (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber, uint32_t Address)

Function description

This function allows to copy LUT from flash to look up RAM.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **FirstLine:** First line enabled on LUT. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- **LinesNumber:** Number of lines enabled on LUT. This parameter must be a number between Min_Data = 1 and Max_Data = 1024.
- **Address:** Start address of LUT in flash.

Return values

- **HAL:** status.

HAL_GFXMMU_DisableLutLines

Function name

HAL_StatusTypeDef HAL_GFXMMU_DisableLutLines (GFXMMU_HandleTypeDef * hgfxmmu, uint32_t FirstLine, uint32_t LinesNumber)

Function description

This function allows to disable a range of LUT lines.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **FirstLine:** First line to disable on LUT. This parameter must be a number between Min_Data = 0 and Max_Data = 1023.
- **LinesNumber:** Number of lines to disable on LUT. This parameter must be a number between Min_Data = 1 and Max_Data = 1024.

Return values

- **HAL:** status.

HAL_GFXMMU_ConfigLutLine

Function name

HAL_StatusTypeDef HAL_GFXMMU_ConfigLutLine (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_LutLineTypeDef * lutLine)

Function description

This function allows to configure one line of LUT.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **lutLine:** LUT line parameters.

Return values

- **HAL:** status.

HAL_GFXMMU_ModifyBuffers

Function name

HAL_StatusTypeDef HAL_GFXMMU_ModifyBuffers (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_BuffersTypeDef * Buffers)

Function description

This function allows to modify physical buffer addresses.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **Buffers:** Buffers parameters.

Return values

- **HAL:** status.

Notes

- This function is only available on STM32U599/STM32U5A9 devices.

HAL_GFXMMU_ModifyAddressCache

Function name

HAL_StatusTypeDef HAL_GFXMMU_ModifyAddressCache (GFXMMU_HandleTypeDef * hgfxmmu, GFXMMU_AddressCacheTypeDef * AddressCache)

Function description

This function allows to modify address cache parameters.

Parameters

- **hgfxmmu:** GFXMMU handle.
- **AddressCache:** address cache parameters.

Return values

- **HAL:** status.

Notes

- This function is only available on STM32U5F9/STM32U5G9 devices.

HAL_GFXMMU_IRQHandler

Function name

void HAL_GFXMMU_IRQHandler (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

This function handles the GFXMMU interrupts.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_ErrorCallback

Function name

void HAL_GFXMMU_ErrorCallback (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

Error callback.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **None.:**

HAL_GFXMMU_GetState

Function name

HAL_GFXMMU_StateTypeDef HAL_GFXMMU_GetState (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

This function allows to get the current GFXMMU handle state.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **GFXMMU:** state.

HAL_GFXMMU_GetError

Function name

uint32_t HAL_GFXMMU_GetError (GFXMMU_HandleTypeDef * hgfxmmu)

Function description

This function allows to get the current GFXMMU error code.

Parameters

- **hgfxmmu:** GFXMMU handle.

Return values

- **GFXMMU:** error code.

29.3 GFXMMU Firmware driver defines

The following section lists the various define and macros of the module.

29.3.1 GFXMMU

GFXMMU

GFXMMU address cache lock buffer

GFXMMU_ADDRESSCACHE_LOCK_BUFFER0

Address Cache locked to buffer 0

GFXMMU_ADDRESSCACHE_LOCK_BUFFER1

Address Cache locked to buffer 1

GFXMMU_ADDRESSCACHE_LOCK_BUFFER2

Address Cache locked to buffer 2

GFXMMU_ADDRESSCACHE_LOCK_BUFFER3

Address Cache locked to buffer 3

GFXMMU blocks per line

GFXMMU_256BLOCKS

256 blocks of 16 bytes per line

GFXMMU_192BLOCKS

192 blocks of 16 bytes per line

GFXMMU cache force

GFXMMU_CACHE_FORCE_DISABLE

Caching not forced

GFXMMU_CACHE_FORCE_ENABLE

Caching forced

GFXMMU cache force parameter

GFXMMU_CACHE_FORCE_FLUSH

Force cache flush

GFXMMU_CACHE_FORCE_INVALIDATE

Force cache invalidate

GFXMMU cache lock

GFXMMU_CACHE_LOCK_DISABLE

Cache not locked to a buffer

GFXMMU_CACHE_LOCK_ENABLE

Cache locked to a buffer

GFXMMU cache lock buffer

GFXMMU_CACHE_LOCK_BUFFER0

Cache locked to buffer 0

GFXMMU_CACHE_LOCK_BUFFER1

Cache locked to buffer 1

GFXMMU_CACHE_LOCK_BUFFER2

Cache locked to buffer 2

GFXMMU_CACHE_LOCK_BUFFER3

Cache locked to buffer 3

GFXMMU Error Code

GFXMMU_ERROR_NONE

No error.

GFXMMU_ERROR_BUFFER0_OVERFLOW
Buffer 0 overflow.

GFXMMU_ERROR_BUFFER1_OVERFLOW
Buffer 1 overflow.

GFXMMU_ERROR_BUFFER2_OVERFLOW
Buffer 2 overflow.

GFXMMU_ERROR_BUFFER3_OVERFLOW
Buffer 3 overflow.

GFXMMU_ERROR_AHB_MASTER
AHB master error.

GFXMMU_ERROR_INVALID_CALLBACK
Invalid callback error.

GFXMMU Exported Macros

__HAL_GFXMMU_RESET_HANDLE_STATE

Description:
- Reset GFXMMU handle state.

Parameters:
- __HANDLE__: GFXMMU handle.

Return value:
- None

GFXMMU interrupts

GFXMMU_AHB_MASTER_ERROR_IT
AHB master error interrupt.

GFXMMU_BUFFER0_OVERFLOW_IT
Buffer 0 overflow interrupt.

GFXMMU_BUFFER1_OVERFLOW_IT
Buffer 1 overflow interrupt.

GFXMMU_BUFFER2_OVERFLOW_IT
Buffer 2 overflow interrupt.

GFXMMU_BUFFER3_OVERFLOW_IT
Buffer 3 overflow interrupt.

GFXMMU LUT line status

GFXMMU_LUT_LINE_DISABLE
LUT line disabled.

GFXMMU_LUT_LINE_ENABLE
LUT line enabled.

GFXMMU outer bufferability

GFXMMU_OUTTER_BUFFERABILITY_DISABLE
No bufferable

GFXMMU_OUTTER_BUFFERABILITY_ENABLE

Bufferable

GFXMMU outer cachability

GFXMMU_OUTTER_CACHABILITY_DISABLE

No cacheable

GFXMMU_OUTTER_CACHABILITY_ENABLE

Cacheable

GFXMMU pre-fetch

GFXMMU_PREFETCH_DISABLE

Pre-fetch disable

GFXMMU_PREFETCH_ENABLE

Pre-fetch enable

30 HAL GFXTIM Generic Driver

30.1 GFXTIM Firmware driver registers structures

30.1.1 GFXTIM_InitTypeDef

GFXTIM_InitTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- ***uint32_t SynchroSrc***
- ***uint32_t TearingEffectSrc***
- ***uint32_t TearingEffectPolarity***
- ***uint32_t TearingEffectInterrupt***

Field Documentation

- ***uint32_t GFXTIM_InitTypeDef::SynchroSrc***
  Synchronization signals (HSYNC and VSYNC) sources. This parameter can be a value of ***GFXTIM_SynchroSrc***
- ***uint32_t GFXTIM_InitTypeDef::TearingEffectSrc***
  Tearing effect source This parameter can be a value of ***GFXTIM_TearingEffectSrc***
- ***uint32_t GFXTIM_InitTypeDef::TearingEffectPolarity***
  Tearing effect source This parameter can be a value of ***GFXTIM_TearingEffectPolarity***
- ***uint32_t GFXTIM_InitTypeDef::TearingEffectInterrupt***
  Tearing effect interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

30.1.2 __GFXTIM_HandleTypeDef

__GFXTIM_HandleTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- ***GFXTIM_TypeDef * Instance***
- ***__IO HAL_GFXTIM_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***GFXTIM_InitTypeDef Init***
- ***void(* HAL_GFXTIM_AbsoluteTimer_AFCC1Callback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_AFCOFCallback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_ALCC1Callback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_ALCC2Callback***
- ***void(* HAL_GFXTIM_AbsoluteTimer_ALCOFCallback***
- ***void(* HAL_GFXTIM_RelativeTimer_RFC1RCallback***
- ***void(* HAL_GFXTIM_RelativeTimer_RFC2RCallback***
- ***void(* HAL_GFXTIM_TECallback***
- ***void(* HAL_GFXTIM_EventGenerator_EV1Callback***
- ***void(* HAL_GFXTIM_EventGenerator_EV2Callback***
- ***void(* HAL_GFXTIM_EventGenerator_EV3Callback***
- ***void(* HAL_GFXTIM_EventGenerator_EV4Callback***
- ***void(* HAL_GFXTIM_WatchdogTimer_AlarmCallback***
- ***void(* HAL_GFXTIM_WatchdogTimer_PreAlarmCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***GFXTIM_TypeDef* __GFXTIM_HandleTypeDef::Instance***
  GFXTIM instance

- ***__IO HAL_GFXTIM_StateTypeDef __GFXTIM_HandleTypeDef::State***
  GFXTIM state
- ***__IO uint32_t __GFXTIM_HandleTypeDef::ErrorCode***
  GFXTIM error code
- ***GFXTIM_InitTypeDef __GFXTIM_HandleTypeDef::Init***
  GFXTIM initialization
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_AFCC1Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute frame counter compare 1 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_AFCOFCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute frame counter overflow callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_ALCC1Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute line counter compare 1 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_ALCC2Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute line counter compare 2 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_AbsoluteTimer_ALCOFCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Absolute line counter overflow callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_RelativeTimer_RFC1RCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Relative frame counter 1 reload callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_RelativeTimer_RFC2RCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Relative frame counter 2 reload callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_TECallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Tearing effect callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV1Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 1 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV2Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 2 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV3Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 3 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_EventGenerator_EV4Callback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Event events 4 callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_WatchdogTimer_AlarmCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Watchdog alarm callback
- ***void(* __GFXTIM_HandleTypeDef::HAL_GFXTIM_WatchdogTimer_PreAlarmCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM Watchdog pre alarm callback
- ***void(* __GFXTIM_HandleTypeDef::ErrorCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM error callback
- ***void(* __GFXTIM_HandleTypeDef::MspInitCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM MSP initialization user callback
- ***void(* __GFXTIM_HandleTypeDef::MspDeInitCallback)(struct __GFXTIM_HandleTypeDef *hgfxtim)***
  GFXTIM MSP de-initialization user callback

30.1.3 GFXTIM_ClockGeneratorConfigTypeDef

GFXTIM_ClockGeneratorConfigTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- *uint32_t LCCHwReloadSrc*
- *uint32_t LCCReloadValue*
- *uint32_t LCCClockSrc*
- *uint32_t LineClockSrc*
- *uint32_t FCCHwReloadSrc*
- *uint32_t FCCReloadValue*
- *uint32_t FCCClockSrc*
- *uint32_t FrameClockSrc*
- *uint32_t LineClockCalib*
- *uint32_t FrameClockCalib*

Field Documentation

- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LCCHwReloadSrc***
  Line Clock Counter hardware reload source This parameter can be a value of ***GFXTIM_LCCHwReloadSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LCCReloadValue***
  Line Clock Counter reload value (22 bits) This parameter must be a number between Min_Data = 1 and Max_Data = 4194303
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LCCClockSrc***
  Line Clock Counter Clock Source This parameter can be a value of ***GFXTIM_LCCClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LineClockSrc***
  Line Clock Source This parameter can be a value of ***GFXTIM_LineClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FCCHwReloadSrc***
  Frame Clock Counter hardware reload source This parameter can be a value of ***GFXTIM_FCCHwReloadSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FCCReloadValue***
  Frame Clock Counter reload value (12 bits) This parameter must be a number between Min_Data = 1 and Max_Data = 4095
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FCCClockSrc***
  Frame Clock Counter Clock Source This parameter can be a value of ***GFXTIM_FCCClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FrameClockSrc***
  Frame Clock Source This parameter can be a value of ***GFXTIM_FrameClockSrc***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::LineClockCalib***
  Debug purpose This parameter can be a value of ***GFXTIM_LineClockCalib***
- ***uint32_t GFXTIM_ClockGeneratorConfigTypeDef::FrameClockCalib***
  Debug purpose This parameter can be a value of ***GFXTIM_FrameClockCalib***

30.1.4 GFXTIM_AbsoluteTimerConfigTypeDef

GFXTIM_AbsoluteTimerConfigTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- *uint32_t FrameCompare1Value*
- *uint32_t FrameCounterValue*
- *uint32_t FrameOverflowInterrupt*
- *uint32_t FrameCompare1Interrupt*
- *uint32_t LineCompare1Value*
- *uint32_t LineCompare2Value*
- *uint32_t LineCounterValue*
- *uint32_t LineOverflowInterrupt*
- *uint32_t LineCompare1Interrupt*
- *uint32_t LineCompare2Interrupt*

Field Documentation

- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameCompare1Value***
  Absolute Frame Compare 1 value (20 bits) This parameter must be a number between 1 and 1048575

- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameCounterValue***
  Absolute Frame Counter initial value (20 bits) This parameter must be a number between 1 and 1048575
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameOverflowInterrupt***
  Absolute Frame Counter Overflow Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::FrameCompare1Interrupt***
  Absolute Frame Compare 1 Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare1Value***
  Absolute Line Compare 1 value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare2Value***
  Absolute Line Compare 2 value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCounterValue***
  Absolute Line Counter value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineOverflowInterrupt***
  Absolute Line Counter Overflow Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare1Interrupt***
  Absolute Line Compare 1 Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_AbsoluteTimerConfigTypeDef::LineCompare2Interrupt***
  Absolute Line Compare 2 Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

30.1.5 GFXTIM_RelativeTimerConfigTypeDef

GFXTIM_RelativeTimerConfigTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- ***uint32_t AutoReloadValue***
- ***uint32_t CounterMode***
- ***uint32_t ReloadInterrupt***

Field Documentation

- ***uint32_t GFXTIM_RelativeTimerConfigTypeDef::AutoReloadValue***
  Auto reload value (12 bits) This parameter must be a number between 1 and 4095
- ***uint32_t GFXTIM_RelativeTimerConfigTypeDef::CounterMode***
  Counter Mode This parameter can be a value of GFXTIM_RelativeCounterMode
- ***uint32_t GFXTIM_RelativeTimerConfigTypeDef::ReloadInterrupt***
  Relative Frame Counter Reload Interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

30.1.6 GFXTIM_EventGeneratorConfigTypeDef

GFXTIM_EventGeneratorConfigTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- ***uint32_t LineEvent***
- ***uint32_t FrameEvent***
- ***uint32_t EventInterrupt***

Field Documentation

- ***uint32_t GFXTIM_EventGeneratorConfigTypeDef::LineEvent***
  Line event selection This parameter can be a value of GFXTIM_EventLine
- ***uint32_t GFXTIM_EventGeneratorConfigTypeDef::FrameEvent***
  Frmae event selection This parameter can be a value of GFXTIM_EventFrame
- ***uint32_t GFXTIM_EventGeneratorConfigTypeDef::EventInterrupt***
  Event interrupt Enable or Disable This parameter can be a value of ***GFXTIM_Interrupt***

30.1.7 GFXTIM_WatchdogConfigTypeDef

GFXTIM_WatchdogConfigTypeDef is defined in the stm32u5xx_hal_gfxtim.h

Data Fields

- ***uint32_t ClockSrc***
- ***uint32_t AutoReloadValue***
- ***uint32_t HwReloadConfig***
- ***uint32_t PreAlarmValue***
- ***uint32_t AlarmInterrupt***
- ***uint32_t PreAlarmInterrupt***

Field Documentation

- ***uint32_t GFXTIM_WatchdogConfigTypeDef::ClockSrc***
  Clock source This parameter can be a value of GFXTIM_WatchdogClockSrc
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::AutoReloadValue***
  Reload value (16 bits) This parameter must be a number between 1 and 65535
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::HwReloadConfig***
  Hardware reload configuration This parameter can be a value of GFXTIM_WatchdogHwReloadConfig
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::PreAlarmValue***
  Pre-alarm value (16 bits) This parameter must be a number between 1 and 65535
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::AlarmInterrupt***
  Interrupt Enable or Disable when watchdog counter reaches 0 This parameter can be a value of ***GFXTIM_Interrupt***
- ***uint32_t GFXTIM_WatchdogConfigTypeDef::PreAlarmInterrupt***
  Interrupt Enable or Disable when watchdog counter reaches pre-alarm value This parameter can be a value of ***GFXTIM_Interrupt***

30.2 GFXTIM Firmware driver API description

The following section lists the various functions of the GFXTIM library.

30.2.1 How to use this driver

Initialization and de-initialization

1. User has first to initialize GFXTIM.
2. As prerequisite, fill in the HAL_GFXTIM_MspInit() :
   – Enable GFXTIM with __HAL_RCC_GFXTIM_CLK_ENABLE
   – Enable the clocks for the used GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   – Configure these pins in alternate mode using HAL_GPIO_Init().
   – If interrupt mode is used, enable and configure GFXTIM interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
1. User can de-initialize GFXTIM with HAL_GFXTIM_DeInit() function.

generic functions

1. HAL_GFXTIM_IRQHandler will be called when GFXTIM interrupt occurs.
2. HAL_GFXTIM_ErrorCallback will be called when GFXTIM or ADF error occurs.
3. Use HAL_GFXTIM_GetState() to get the current GFXTIM or ADF instance state.
4. Use HAL_GFXTIM_GetErrorCode() to get the current GFXTIM or ADF instance error code. #if defined(GENERATOR_CALLBACKS_REGISTERING_AVAILABLE)

Callback registration

30.2.2 Initialization and de-initialization functions

This section provides functions allowing to :

- Initialize the GFXTIM instance.
- De-initialize the GFXTIM instance.
- Register and unregister callbacks.

This section contains the following APIs:

- *HAL_GFXTIM_Init()*
- *HAL_GFXTIM_DeInit()*
- *HAL_GFXTIM_MspInit()*
- *HAL_GFXTIM_MspDeInit()*
- *HAL_GFXTIM_RegisterCallback()*
- *HAL_GFXTIM_UnRegisterCallback()*
- *HAL_GFXTIM_TECallback()*

30.2.3 Clock Generator functions

This section provides functions allowing to :

- Configure the clock generator.
- Force reload of FCC and LCC.

This section contains the following APIs:

- *HAL_GFXTIM_ClockGenerator_Config()*
- *HAL_GFXTIM_ClockGenerator_Reload()*
- *HAL_GFXTIM_TECallback()*

30.2.4 Absolute Timers functions

This section provides functions allowing to :

- Configure the absolute timer.
- Start the absolute timer.
- Stop the absolute timer.
- Reset the absolute timer counters.
- Get the absolute time value.
- Set the absolute frame compare value.
- Set the absolute line compare value.

This section contains the following APIs:

- *HAL_GFXTIM_AbsoluteTimer_Config()*
- *HAL_GFXTIM_AbsoluteTimer_Start()*
- *HAL_GFXTIM_AbsoluteTimer_Stop()*
- *HAL_GFXTIM_AbsoluteTimer_Reset()*
- *HAL_GFXTIM_AbsoluteTimer_GetCounter()*
- *HAL_GFXTIM_AbsoluteTimer_SetFrameCompare()*
- *HAL_GFXTIM_AbsoluteTimer_SetLineCompare()*
- *HAL_GFXTIM_AbsoluteTimer_AFCOFCallback()*
- *HAL_GFXTIM_AbsoluteTimer_AFCC1Callback()*
- *HAL_GFXTIM_AbsoluteTimer_ALCC1Callback()*
- *HAL_GFXTIM_AbsoluteTimer_ALCC2Callback()*
- *HAL_GFXTIM_AbsoluteTimer_ALCOFCallback()*

30.2.5 Relative Timer functions

This section provides functions allowing to :

- Configure a relative timer.
- Start a relative timer counter.
- Stop a relative timer counter.
- Force a relative timer counter reload.
- Set a relative timer reload value.
- Get a relative timer counter value.

This section contains the following APIs:

- *HAL_GFXTIM_RelativeTimer_Config()*
- *HAL_GFXTIM_RelativeTimer_Start()*
- *HAL_GFXTIM_RelativeTimer_Stop()*
- *HAL_GFXTIM_RelativeTimer_ForceReload()*
- *HAL_GFXTIM_RelativeTimer_SetReload()*
- *HAL_GFXTIM_RelativeTimer_GetCounter()*
- *HAL_GFXTIM_RelativeTimer_RFC1RCallback()*
- *HAL_GFXTIM_RelativeTimer_RFC2RCallback()*

30.2.6 Event Generator functions

This section provides functions allowing to :
- Configure an Event Generator.
- Enable an Event Generator.
- Disable an Event Generator.

This section contains the following APIs:
- *HAL_GFXTIM_EventGenerator_Config()*
- *HAL_GFXTIM_EventGenerator_Enable()*
- *HAL_GFXTIM_EventGenerator_Disable()*
- *HAL_GFXTIM_EventGenerator_EV1Callback()*
- *HAL_GFXTIM_EventGenerator_EV2Callback()*
- *HAL_GFXTIM_EventGenerator_EV3Callback()*
- *HAL_GFXTIM_EventGenerator_EV4Callback()*

30.2.7 Watchdog functions

This section provides functions allowing to :
- Configure the Watchdog.
- Enable the Watchdog
- Disable the Watchdog.
- Refresh the Watchdog.

This section contains the following APIs:
- *HAL_GFXTIM_WatchdogTimer_Config()*
- *HAL_GFXTIM_WatchdogTimer_Enable()*
- *HAL_GFXTIM_WatchdogTimer_Disable()*
- *HAL_GFXTIM_WatchdogTimer_Refresh()*
- *HAL_GFXTIM_WatchdogTimer_AlarmCallback()*
- *HAL_GFXTIM_WatchdogTimer_PreAlarmCallback()*

30.2.8 Generic functions

This section provides functions allowing to :
- Handle GFXTIM interrupt.
- Inform user that error occurs.
- Get the current GFXTIM instance state
- Get the current GFXTIM instance error code.

This section contains the following APIs:
- *HAL_GFXTIM_IRQHandler()*
- *HAL_GFXTIM_ErrorCallback()*
- *HAL_GFXTIM_GetState()*
- *HAL_GFXTIM_GetError()*

30.2.9 Detailed description of functions

HAL_GFXTIM_Init

Function name

HAL_StatusTypeDef HAL_GFXTIM_Init (GFXTIM_HandleTypeDef * hgfxtim)

Function description

Initialize the GFXTIM instance according to the specified parameters in the GFXTIM_InitTypeDef structure and initialize the associated handle.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_DeInit

Function name

HAL_StatusTypeDef HAL_GFXTIM_DeInit (GFXTIM_HandleTypeDef * hgfxtim)

Function description

De-initialize the GFXTIM instance.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_MspInit

Function name

void HAL_GFXTIM_MspInit (GFXTIM_HandleTypeDef * hgfxtim)

Function description

Initialize the GFXTIM instance MSP.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_MspDeInit

Function name

void HAL_GFXTIM_MspDeInit (GFXTIM_HandleTypeDef * hgfxtim)

Function description

De-initialize the GFXTIM instance MSP.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXTIM_RegisterCallback (GFXTIM_HandleTypeDef * hgfxtim, HAL_GFXTIM_CallbackIDTypeDef CallbackID, pGFXTIM_CallbackTypeDef pCallback)

Function description

Register a user GFXTIM callback to be used instead of the weak predefined callback.

Parameters

- **hgfxtim:** GFXTIM handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_GFXTIM_AFC_COMPARE1_CB_ID Absolute frame counter compare 1 callback ID
  - HAL_GFXTIM_AFC_OVERFLOW_CB_ID Absolute frame counter overflow callback ID
  - HAL_GFXTIM_ALC_COMPARE1_CB_ID Absolute line counter compare 1 callback ID
  - HAL_GFXTIM_ALC_COMPARE2_CB_ID Absolute line counter compare 2 callback ID
  - HAL_GFXTIM_ALC_OVERFLOW_CB_ID Absolute line counter overflow callback ID
  - HAL_GFXTIM_RFC1_RELOAD_CB_ID Relative frame counter 1 reload callback ID
  - HAL_GFXTIM_RFC2_RELOAD_CB_ID Relative frame counter 2 reload callback ID
  - HAL_GFXTIM_TE_CB_ID External tearing effect callback ID
  - HAL_GFXTIM_EVENT1_CB_ID Event events 1 callback ID
  - HAL_GFXTIM_EVENT2_CB_ID Event events 2 callback ID
  - HAL_GFXTIM_EVENT3_CB_ID Event events 3 callback ID
  - HAL_GFXTIM_EVENT4_CB_ID Event events 4 callback ID
  - HAL_GFXTIM_WDG_ALARM_CB_ID Watchdog alarm callback ID
  - HAL_GFXTIM_WDG_PREALARM_CB_ID Watchdog pre alarm callback ID
  - HAL_GFXTIM_ERROR_CB_ID error callback ID
  - HAL_GFXTIM_MSP_INIT_CB_ID MSP initialization user callback ID
  - HAL_GFXTIM_MSP_DEINIT_CB_ID MSP de-initialization user callback ID
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_GFXTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_GFXTIM_UnRegisterCallback (GFXTIM_HandleTypeDef * hgfxtim, HAL_GFXTIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a user GFXTIM callback.

Parameters

- **hgfxtim:** GFXTIM handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  – HAL_GFXTIM_AFC_COMPARE1_CB_ID Absolute frame counter compare 1 callback ID
  – HAL_GFXTIM_AFC_OVERFLOW_CB_ID Absolute frame counter overflow callback ID
  – HAL_GFXTIM_ALC_COMPARE1_CB_ID Absolute line counter compare 1 callback ID
  – HAL_GFXTIM_ALC_COMPARE2_CB_ID Absolute line counter compare 2 callback ID
  – HAL_GFXTIM_ALC_OVERFLOW_CB_ID Absolute line counter overflow callback ID
  – HAL_GFXTIM_RFC1_RELOAD_CB_ID Relative frame counter 1 reload callback ID
  – HAL_GFXTIM_RFC2_RELOAD_CB_ID Relative frame counter 2 reload callback ID
  – HAL_GFXTIM_TE_CB_ID External tearing effect callback ID
  – HAL_GFXTIM_EVENT1_CB_ID Event events 1 callback ID
  – HAL_GFXTIM_EVENT2_CB_ID Event events 2 callback ID
  – HAL_GFXTIM_EVENT3_CB_ID Event events 3 callback ID
  – HAL_GFXTIM_EVENT4_CB_ID Event events 4 callback ID
  – HAL_GFXTIM_WDG_ALARM_CB_ID Watchdog alarm callback ID
  – HAL_GFXTIM_WDG_PREALARM_CB_ID Watchdog pre alarm callback ID
  – HAL_GFXTIM_ERROR_CB_ID error callback ID
  – HAL_GFXTIM_MSP_INIT_CB_ID MSP initialization user callback ID
  – HAL_GFXTIM_MSP_DEINIT_CB_ID MSP de-initialization user callback ID

Return values

- **HAL:** status.

HAL_GFXTIM_TECallback

Function name

void HAL_GFXTIM_TECallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Tearing effect callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_ClockGenerator_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_ClockGenerator_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_ClockGeneratorConfigTypeDef * pClockGeneratorConfig)

Function description

This function configures the clock generator.

Parameters

- **hgfxtim:** GFXTIM handle.
- **pClockGeneratorConfig:** Clock Generator configuration parameters.

Return values

- **HAL:** status.

HAL_GFXTIM_ClockGenerator_Reload

Function name

HAL_StatusTypeDef HAL_GFXTIM_ClockGenerator_Reload (GFXTIM_HandleTypeDef * hgfxtim, uint32_t ClockGeneratorCounter)

Function description

This function force clock generator counter(s) reload.

Parameters

- **hgfxtim:** GFXTIM handle.
- **ClockGeneratorCounter:** Clock Generator counter This parameter can be a value of GFXTIM Clock Generator Counter.

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_AbsoluteTimerConfigTypeDef * pAbsoluteTimerConfig)

Function description

This function configures an absolute Timer.

Parameters

- **hgfxtim:** GFXTIM handle.
- **pAbsoluteTimerConfig:** pointer to a GFXTIM_AbsoluteTimerConfigTypeDef structure that contains absoluite timer comparators and counters values.

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_Start

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Start (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function starts absolute timer.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_Stop

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Stop (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function stops absolute timer counter(s).

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_Reset

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_Reset (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function resets absolute timer counters.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_GetCounter

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_GetCounter (const GFXTIM_HandleTypeDef * hgfxtim, uint32_t AbsoluteTime, uint32_t * pValue)

Function description

This function gets absolute timer value.

Parameters

- **hgfxtim:** GFXTIM handle.
- **AbsoluteTime:** absolute time This parameter can be a value of GFXTIM Absolute Time.
- **pValue:** Absolute time value

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_SetFrameCompare

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_SetFrameCompare (GFXTIM_HandleTypeDef * hgfxtim, uint32_t Value)

Function description

This function sets absolute frame compare value.

Parameters

- **hgfxtim:** GFXTIM handle.
- **Value:** Absolute frame compare 1 value This parameter can be a number between Min_Data = 0x00000 and Max_Data = 0xFFFFF

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_SetLineCompare

Function name

HAL_StatusTypeDef HAL_GFXTIM_AbsoluteTimer_SetLineCompare (GFXTIM_HandleTypeDef * hgfxtim, uint32_t AbsoluteLineComparator, uint32_t Value)

Function description

This function sets line compare value.

Parameters

- **hgfxtim:** GFXTIM handle.
- **AbsoluteLineComparator:** Absolute line compare value This parameter can be a value of GFXTIM Absolute Line Comparator.
- **Value:** Absolute line compare value This parameter can be a number between Min_Data = 0x000 and Max_Data = 0xFFF

Return values

- **HAL:** status.

HAL_GFXTIM_AbsoluteTimer_AFCC1Callback

Function name

void HAL_GFXTIM_AbsoluteTimer_AFCC1Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Absolute frame counter compare 1 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_AbsoluteTimer_AFCOFCallback

Function name

void HAL_GFXTIM_AbsoluteTimer_AFCOFCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Absolute frame counter overflow callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_AbsoluteTimer_ALCC1Callback

Function name

void HAL_GFXTIM_AbsoluteTimer_ALCC1Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Absolute line counter compare 1 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_AbsoluteTimer_ALCC2Callback

Function name

void HAL_GFXTIM_AbsoluteTimer_ALCC2Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Absolute line counter compare 2 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_AbsoluteTimer_ALCOFCallback

Function name

void HAL_GFXTIM_AbsoluteTimer_ALCOFCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Absolute line counter overflow callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_RelativeTimer_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_RelativeTimerConfigTypeDef * pRelativeTimerConfig, uint32_t RelativeTimer)

Function description

This function configures a Relative Timer.

Parameters

- **hgfxtim:** GFXTIM handle.
- **pRelativeTimerConfig:** pointer to a GFXTIM_RelativeTimerConfigTypeDef structure that contains relative timer comparators and counters values.
- **RelativeTimer:** Relative Timer identifier This parameter can be a value of GFXTIM Relative Timer.

Return values

- **HAL:** status.

HAL_GFXTIM_RelativeTimer_Start

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_Start (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer)

Function description

This function starts a relative Timer.

Parameters

- **hgfxtim:** GFXTIM handle.
- **RelativeTimer:** Relative timer counter to Enable This parameter can be a value of GFXTIM Relative Timer.

Return values

- **HAL:** status.

HAL_GFXTIM_RelativeTimer_Stop

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_Stop (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer)

Function description

This function stops a relative Timer counter.

Parameters

- **hgfxtim:** GFXTIM handle.
- **RelativeTimer:** Relative timer counter to Disable This parameter can be a value of GFXTIM Relative Timer.

Return values

- **HAL:** status.

HAL_GFXTIM_RelativeTimer_ForceReload

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_ForceReload (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer)

Function description

This function force a relative Timer reload.

Parameters

- **hgfxtim:** GFXTIM handle.
- **RelativeTimer:** Relative timer to Foce Reload This parameter can be a value of GFXTIM Relative Timer.

Return values

- **HAL:** status.

HAL_GFXTIM_RelativeTimer_SetReload

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_SetReload (GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer, uint32_t Value)

Function description

This function sets Relative frame timer reload value.

Parameters

- **hgfxtim:** GFXTIM handle.
- **RelativeTimer:** specifies the Auto-reload register to be modified. This parameter can be a value of GFXTIM Relative Timer.
- **Value:** Reload value This parameter can be a number between Min_Data = 0x000 and Max_Data = 0xFFF

Return values

- **HAL:** status.

HAL_GFXTIM_RelativeTimer_GetCounter

Function name

HAL_StatusTypeDef HAL_GFXTIM_RelativeTimer_GetCounter (const GFXTIM_HandleTypeDef * hgfxtim, uint32_t RelativeTimer, uint32_t * pValue)

Function description

This function sets Relative frame timer compare value.

Parameters

- **hgfxtim:** GFXTIM handle.
- **RelativeTimer:** Relative frame counter reload This parameter can be a value of GFXTIM Relative Timer.
- **pValue:** pointer to a relative frame counter value

Return values

- **HAL:** status.

HAL_GFXTIM_RelativeTimer_RFC1RCallback

Function name

void HAL_GFXTIM_RelativeTimer_RFC1RCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Relative frame counter 1 reload callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_RelativeTimer_RFC2RCallback

Function name

void HAL_GFXTIM_RelativeTimer_RFC2RCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Relative frame counter 2 reload callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_EventGenerator_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_EventGenerator_Config (GFXTIM_HandleTypeDef * hgfxtim, uint32_t EventGenerator, const GFXTIM_EventGeneratorConfigTypeDef * pEventGeneratorConfig)

Function description

This function configures an Event Generator.

Parameters

- **hgfxtim:** GFXTIM handle.
- **EventGenerator:** Event Generator This parameter can be a value of GFXTIM Event Generator ID.
- **pEventGeneratorConfig:** pointer to a GFXTIM_EventGeneratorConfigTypeDef structure that contains Event Generator configuration parameters.

Return values

- **HAL:** status.

HAL_GFXTIM_EventGenerator_Enable

Function name

HAL_StatusTypeDef HAL_GFXTIM_EventGenerator_Enable (GFXTIM_HandleTypeDef * hgfxtim, uint32_t EventGenerator)

Function description

This function enables an Event Generator.

Parameters

- **hgfxtim:** GFXTIM handle.
- **EventGenerator:** Event Generator This parameter can be a value of GFXTIM Event Generator ID.

Return values

- **HAL:** status.

HAL_GFXTIM_EventGenerator_Disable

Function name

HAL_StatusTypeDef HAL_GFXTIM_EventGenerator_Disable (GFXTIM_HandleTypeDef * hgfxtim, uint32_t EventGenerator)

Function description

This function disables an Event Generator.

Parameters

- **hgfxtim:** GFXTIM handle.
- **EventGenerator:** Event Generator This parameter can be a value of GFXTIM Event Generator ID.

Return values

- **HAL:** status.

HAL_GFXTIM_EventGenerator_EV1Callback

Function name

void HAL_GFXTIM_EventGenerator_EV1Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Combined events 1 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_EventGenerator_EV2Callback

Function name

void HAL_GFXTIM_EventGenerator_EV2Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Combined events 2 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_EventGenerator_EV3Callback

Function name

void HAL_GFXTIM_EventGenerator_EV3Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Combined events 3 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_EventGenerator_EV4Callback

Function name

void HAL_GFXTIM_EventGenerator_EV4Callback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Combined events 4 callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_WatchdogTimer_Config

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Config (GFXTIM_HandleTypeDef * hgfxtim, const GFXTIM_WatchdogConfigTypeDef * pWatchdogConfig)

Function description

This function configures the Watchdog.

Parameters

- **hgfxtim:** GFXTIM handle.
- **pWatchdogConfig:** Watchdog configuration parameters.

Return values

- **HAL:** status.

HAL_GFXTIM_WatchdogTimer_Enable

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Enable (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function Enable the Watchdog Counter.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_WatchdogTimer_Disable

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Disable (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function Disable the Watchdog Counter.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_WatchdogTimer_Refresh

Function name

HAL_StatusTypeDef HAL_GFXTIM_WatchdogTimer_Refresh (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function refresh the Watchdog counter.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **HAL:** status.

HAL_GFXTIM_WatchdogTimer_AlarmCallback

Function name

void HAL_GFXTIM_WatchdogTimer_AlarmCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Watchdog alarm callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_WatchdogTimer_PreAlarmCallback

Function name

void HAL_GFXTIM_WatchdogTimer_PreAlarmCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM Watchdog pre alarm callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_IRQHandler

Function name

void HAL_GFXTIM_IRQHandler (GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function handles the GFXTIM interrupts.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_ErrorCallback

Function name

void HAL_GFXTIM_ErrorCallback (GFXTIM_HandleTypeDef * hgfxtim)

Function description

GFXTIM error callback.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **None.:**

HAL_GFXTIM_GetError

Function name

uint32_t HAL_GFXTIM_GetError (const GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function get the current GFXTIM error.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **GFXTIM:** error code.

HAL_GFXTIM_GetState

Function name

HAL_GFXTIM_StateTypeDef HAL_GFXTIM_GetState (const GFXTIM_HandleTypeDef * hgfxtim)

Function description

This function get the current GFXTIM state.

Parameters

- **hgfxtim:** GFXTIM handle.

Return values

- **GFXTIM:** state.

30.3 GFXTIM Firmware driver defines

The following section lists the various define and macros of the module.

30.3.1 GFXTIM

GFXTIM

GFXTIM Absolute Line Comparator

GFXTIM_ABSOLUTE_LINE_COMPARE1

Absolute line compare 1

GFXTIM_ABSOLUTE_LINE_COMPARE2

Absolute line compare 2

GFXTIM Absolute Time

GFXTIM_ABSOLUTE_GLOBAL_TIME

Absolute global time (frame and line) counters ATR

GFXTIM_ABSOLUTE_FRAME_TIME

Absolute frame counter AFCR

GFXTIM_ABSOLUTE_LINE_TIME

Absolute line counter ALCR

GFXTIM Clock Generator Counter

GFXTIM_LINE_CLK_COUNTER

Line clock counter

GFXTIM_FRAME_CLK_COUNTER

Frame clock counter

GFXTIM Error Code

GFXTIM_ERROR_NONE

No error

GFXTIM_ERROR_STATE

State error

GFXTIM_ERROR_INVALID_CALLBACK

Invalid callback error occurs

GFXTIM Event generator Frame Source selection

GFXTIM_FRAME_EVENT_NONE

None

GFXTIM_FRAME_EVENT_AFC_OVERFLOW

Absolute frame counter overflow

GFXTIM_FRAME_EVENT_AFC_COMPARE

Absolute frame counter compare

GFXTIM_FRAME_EVENT_RFC1_RELOAD

Relative frame counter 1 reload

GFXTIM_FRAME_EVENT_RFC2_RELOAD

Relative frame counter 1 reload

GFXTIM Event Generator ID

GFXTIM_EVENT_GENERATOR_1

Event Generator 1

GFXTIM_EVENT_GENERATOR_2

Event Generator 2

GFXTIM_EVENT_GENERATOR_3

Event Generator 3

GFXTIM_EVENT_GENERATOR_4

Event Generator 4

GFXTIM Event generator Line source selection

GFXTIM_LINE_EVENT_NONE

None

GFXTIM_LINE_EVENT_ALC_OVERFLOW

Absolute line counter overflow

GFXTIM_LINE_EVENT_TE

Tearing effect

GFXTIM_LINE_EVENT_ALC1_COMPARE

Absolute line counter 1 compare

GFXTIM_LINE_EVENT_ALC2_COMPARE

Absolute line counter 2 compare

GFXTIM Exported Macros

__HAL_GFXTIM_RESET_HANDLE_STATE

Description:

- Reset GFXTIM handle state.

Parameters:

- __HANDLE__: GFXTIM handle.

Return value:

- None

__HAL_GFXTIM_GET_FLAG

Description:

- Check whether the specified GFXTIM flag is set or not.

Parameters:

- __HANDLE__: GFXTIM handle
- __FLAG__: GFXTIM flag This parameter can be one or a combination of the following values:
  - GFXTIM_FLAG_AFCO Absolute Frame Counter Overflow Flag
  - GFXTIM_FLAG_ALCO Absolute Line Counter Overflow Flag
  - GFXTIM_FLAG_TE Tearing Effect Flag
  - GFXTIM_FLAG_AFCC1 Absolute Frame Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC1 Absolute Line Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC2 Absolute Line Counter Compare 2 Flag
  - GFXTIM_FLAG_RFC1R Relative Frame Counter 1 Reload Flag
  - GFXTIM_FLAG_RFC2R Relative Frame Counter 2 Reload Flag
  - GFXTIM_FLAG_EV1 Event 1 Flag
  - GFXTIM_FLAG_EV2 Event 2 Flag
  - GFXTIM_FLAG_EV3 Event 3 Flag
  - GFXTIM_FLAG_EV4 Event 4 Flag
  - GFXTIM_FLAG_WDGA Watchdog Alarm Flag
  - GFXTIM_FLAG_WDGP Watchdog Pre-alarm Flag

Return value:

- State: of flag (TRUE or FALSE).

__HAL_GFXTIM_CLEAR_FLAG

Description:

- Clear the specified GFXTIM flag.

Parameters:

- __HANDLE__: GFXTIM handle
- __FLAG__: GFXTIM flag This parameter can be one or a combination of the following values:
  - GFXTIM_FLAG_AFCO Absolute Frame Counter Overflow Flag
  - GFXTIM_FLAG_ALCO Absolute Line Counter Overflow Flag
  - GFXTIM_FLAG_TE Tearing Effect Flag
  - GFXTIM_FLAG_AFCC1 Absolute Frame Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC1 Absolute Line Counter Compare 1 Flag
  - GFXTIM_FLAG_ALCC2 Absolute Line Counter Compare 2 Flag
  - GFXTIM_FLAG_RFC1R Relative Frame Counter 1 Reload Flag
  - GFXTIM_FLAG_RFC2R Relative Frame Counter 2 Reload Flag
  - GFXTIM_FLAG_EV1 Event 1 Flag
  - GFXTIM_FLAG_EV2 Event 2 Flag
  - GFXTIM_FLAG_EV3 Event 3 Flag
  - GFXTIM_FLAG_EV4 Event 4 Flag
  - GFXTIM_FLAG_WDGA Watchdog Alarm Flag
  - GFXTIM_FLAG_WDGP Watchdog Pre-alarm Flag

Return value:

- None

GFXTIM Frame CLock Counter Clock Source

GFXTIM_FCC_CLK_SRC_DISABLE

Disable

GFXTIM_FCC_CLK_SRC_LCC_UNDERFLOW
Line Clock Counter underflow

GFXTIM_FCC_CLK_SRC_HSYNC_RISING
HSYNC rising edge

GFXTIM_FCC_CLK_SRC_HSYNC_FALLING
HSYNC falling edge

GFXTIM_FCC_CLK_SRC_VSYNC_RISING
VSYNC rising edge

GFXTIM_FCC_CLK_SRC_VSYNC_FALLING
VSYNC falling edge

GFXTIM_FCC_CLK_SRC_TE_RISING
TE rising edge

GFXTIM_FCC_CLK_SRC_TE_FALLING
TE falling edge

GFXTIM Frame Clock Counter Hardware Reload source

GFXTIM_FCC_HW_RELOAD_SRC_NONE
No hardware reload

GFXTIM_FCC_HW_RELOAD_SRC_LCC_UNDERFLOW
Line Clock Counter underflow

GFXTIM_FCC_HW_RELOAD_SRC_HSYNC_RISING
HSYNC rising edge

GFXTIM_FCC_HW_RELOAD_SRC_HSYNC_FALLING
HSYNC falling edge

GFXTIM_FCC_HW_RELOAD_SRC_VSYNC_RISING
VSYNC rising edge

GFXTIM_FCC_HW_RELOAD_SRC_VSYNC_FALLING
VSYNC falling edge

GFXTIM_FCC_HW_RELOAD_SRC_TE_RISING
TE rising edge

GFXTIM_FCC_HW_RELOAD_SRC_TE_FALLING
TE falling edge

GFXTIM flags

GFXTIM_FLAG_AFCO

GFXTIM_FLAG_ALCO

GFXTIM_FLAG_TE

GFXTIM_FLAG_AFCC1

GFXTIM_FLAG_ALCC1

GFXTIM_FLAG_ALCC2

GFXTIM_FLAG_RFC1R

GFXTIM_FLAG_RFC2R

GFXTIM_FLAG_EV1

GFXTIM_FLAG_EV2

GFXTIM_FLAG_EV3

GFXTIM_FLAG_EV4

GFXTIM_FLAG_WDGA

GFXTIM_FLAG_WDGP

GFXTIM Frame Clock Calibration Output (for debug purpose)

GFXTIM_FRAME_CLK_CALIB_DISABLE

Frame clock output calibration Disable

GFXTIM_FRAME_CLK_CALIB_ENABLE

Frame clock output calibration Enable

GFXTIM GFXTIM Frame Clock Source

GFXTIM_FRAME_CLK_SRC_LCC_UNDERFLOW

Line Clock Counter underflow

GFXTIM_FRAME_CLK_SRC_FCC_UNDERFLOW

Frame Clock Counter underflow

GFXTIM_FRAME_CLK_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_FRAME_CLK_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_FRAME_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_FRAME_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_FRAME_CLK_SRC_TE_RISING

TE rising edge

GFXTIM_FRAME_CLK_SRC_TE_FALLING

TE falling edge

GFXTIM Interrupt

GFXTIM_IT_DISABLE

gfxtim_interrupt disable

GFXTIM_IT_ENABLE
gfxtim_interrupt enable

GFXTIM Line Clock Counter Clock Source

GFXTIM_LCC_CLK_SRC_DISABLE
Disable line clock counter

GFXTIM_LCC_CLK_SRC_SYSCLOCK
System clock as line clock counter source

GFXTIM Line Clock Counter Hardware Reload Source

GFXTIM_LCC_HW_RELOAD_SRC_NONE
No hardware reload

GFXTIM_LCC_HW_RELOAD_SRC_FCC_UNDERFLOW
FCC underflow

GFXTIM_LCC_HW_RELOAD_SRC_HSYNC_RISING
HSYNC rising

GFXTIM_LCC_HW_RELOAD_SRC_HSYNC_FALLING
HSYNC falling

GFXTIM_LCC_HW_RELOAD_SRC_VSYNC_RISING
VSYNC rising

GFXTIM_LCC_HW_RELOAD_SRC_VSYNC_FALLING
VSYNC falling

GFXTIM_LCC_HW_RELOAD_SRC_TE_RISING
TE rising

GFXTIM_LCC_HW_RELOAD_SRC_TE_FALLING
TE falling

GFXTIM Line Clock Calibration Output

GFXTIM_LINE_CLK_CALIB_DISABLE
Disable Line clock calibration

GFXTIM_LINE_CLK_CALIB_ENABLE
Enable Line clock calibration

GFXTIM Line Clock Source

GFXTIM_LINE_CLK_SRC_LCC_UNDERFLOW
Line Clock Counter underflow

GFXTIM_LINE_CLK_SRC_FCC_UNDERFLOW
Frame Clock Counter underflow

GFXTIM_LINE_CLK_SRC_HSYNC_RISING
HSYNC rising edge

GFXTIM_LINE_CLK_SRC_HSYNC_FALLING
HSYNC falling edge

GFXTIM_LINE_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_LINE_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_LINE_CLK_SRC_TE_RISING

TE rising edge

GFXTIM_LINE_CLK_SRC_TE_FALLING

TE falling edge

GFXTIM Relative Frame Counter Mode

GFXTIM_MODE_ONE_SHOT

Relative Frame Counter One Shot Mode

GFXTIM_MODE_CONTINUOUS

Relative Frame Counter Continuous Mode

GFXTIM Relative Timer

GFXTIM_RELATIVE_TIMER1

Relative Timer 1

GFXTIM_RELATIVE_TIMER2

Relative Timer 2

GFXTIM Synchronization Source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_0

gfxtim_hsync[0] and gfxtim_vsync[0] are used as synchronization source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_1

gfxtim_hsync[1] and gfxtim_vsync[1] are used as synchronization source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_2

gfxtim_hsync[2] and gfxtim_vsync[2] are used as synchronization source

GFXTIM_SYNC_SRC_HSYNC_VSYNC_3

gfxtim_hsync[3] and gfxtim_vsync[3] are used as synchronization source

GFXTIM Tearing Effect Polarity

GFXTIM_TE_RISING_EDGE

Tearing Effect active on rizing edge

GFXTIM_TE_FALLING_EDGE

Tearing Effect active on falling edge

GFXTIM Tearing Effect Source

GFXTIM_TE_SRC_GPIO

Input pad rising

GFXTIM_TE_SRC_ITE

gfxtim_ite rising

GFXTIM_TE_SRC_HSYNC

HSYNC (see SynchroSrc) rising

GFXTIM_TE_SRC_VSYNC

VSYNC (see SynchroSrc) rising

GFXTIM Watchdog clock source

GFXTIM_WATCHDOG_CLK_SRC_LINE_CLK

Line Clock

GFXTIM_WATCHDOG_CLK_SRC_FRAME_CLK

Frame Clock

GFXTIM_WATCHDOG_CLK_SRC_HSYNC_RISING

HSYNC rising edge

GFXTIM_WATCHDOG_CLK_SRC_HSYNC_FALLING

HSYNC falling edge

GFXTIM_WATCHDOG_CLK_SRC_VSYNC_RISING

VSYNC rising edge

GFXTIM_WATCHDOG_CLK_SRC_VSYNC_FALLING

VSYNC falling edge

GFXTIM_WATCHDOG_CLK_SRC_TE_RISING

Tearing Effect rising edge

GFXTIM_WATCHDOG_CLK_SRC_TE_FALLING

Tearing Effect falling edge

GFXTIM_WATCHDOG_CLK_SRC_EVENT_1

Event Generator 1 output

GFXTIM_WATCHDOG_CLK_SRC_EVENT_2

Event Generator 2 output

GFXTIM_WATCHDOG_CLK_SRC_EVENT_3

Event Generator 3 output

GFXTIM_WATCHDOG_CLK_SRC_EVENT_4

Event Generator 4 output

GFXTIM Watchdog hardware reload configuration

GFXTIM_WATCHDOG_HW_RELOAD_DISABLE

Watchdog hardware reload is disable

GFXTIM_WATCHDOG_HW_RELOAD_RISING_EDGE

Watchdog is reload on rising edge of gfxtim_wrld

GFXTIM_WATCHDOG_HW_RELOAD_FALLING_EDGE

Watchdog is reload on falling edge of gfxtim_wrld

31 HAL GPIO Generic Driver

31.1 GPIO Firmware driver registers structures

31.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32u5xx_hal_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Pull***
- ***uint32_t Speed***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be a value of ***GPIO_pins***
- ***uint32_t GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_mode***
- ***uint32_t GPIO_InitTypeDef::Pull***
  Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of ***GPIO_pull***
- ***uint32_t GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_speed***
- ***uint32_t GPIO_InitTypeDef::Alternate***
  Peripheral to be connected to the selected pins This parameter can be a value of ***GPIOEx_Alternate_function_selection***

31.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

31.2.1 GPIO Peripheral features

- Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:
  - Input mode
  - Analog mode
  - Output mode
  - Alternate function mode
  - External interrupt/event lines
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.
- All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.
- In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.
- The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.
- All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

- The external interrupt/event controller consists of up to 39 edge detectors (16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

31.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   – Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   – Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   – In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   – In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   – Analog mode is required when a pin is to be used as ADC channel or DAC output.
   – In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To set the level of several pins and reset level of several other pins in same cycle, use HAL_GPIO_WriteMultipleStatePin().
7. To lock pin configuration until next reset use HAL_GPIO_LockPin().
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
10. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

31.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_GPIO_Init()*
- *HAL_GPIO_DeInit()*

31.2.4 IO attributes functions

This section contains the following APIs:

- *HAL_GPIO_ConfigPinAttributes()*
- *HAL_GPIO_GetConfigPinAttributes()*

31.2.5 Detailed description of functions

HAL_GPIO_Init

Function name

void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, const GPIO_InitTypeDef * pGPIO_Init)

Function description

Initialize the GPIOx/LPGPIOx peripheral according to the specified parameters in the pGPIO_Init.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the corresponding peripheral for STM32U5 family
- **pGPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

Notes

- If GPIOx/LPGPIOx peripheral pin is used in EXTI_MODE and the pin is secure in case the system implements the security (TZEN=1), it is up to the secure application to insure that the corresponding EXTI line is set secure.

HAL_GPIO_DeInit

Function name

void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)

Function description

De-initialize the GPIOx peripheral registers to their default reset values.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

GPIO_PinState HAL_GPIO_ReadPin (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Read the specified input port pin.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Set or clear the selected data port bit.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
- **PinState:** specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values:
  – GPIO_PIN_RESET: to clear the port pin
  – GPIO_PIN_SET: to set the port pin

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR/LPGPIOx_BSRR and GPIOx_BRR/LPGPIOx_BRR registers to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

HAL_GPIO_WriteMultipleStatePin

Function name

void HAL_GPIO_WriteMultipleStatePin (GPIO_TypeDef * GPIOx, uint16_t PinReset, uint16_t PinSet)

Function description

Set and clear several pins of a dedicated port in same cycle.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **PinReset:** specifies the port bits to be reset This parameter can be any combination of GPIO_Pin_x where x can be (0..15) or zero.
- **PinSet:** specifies the port bits to be set This parameter can be any combination of GPIO_Pin_x where x can be (0..15) or zero.

Return values

- **None:**

Notes

- This function uses GPIOx_BSRR and GPIOx_BRR registers to allow atomic read/modify accesses.
- Both PinReset and PinSet combinations shall not get any common bit, else assert would be triggered.
- At least one of the two parameters used to set or reset shall be different from zero.

HAL_GPIO_TogglePin

Function name

void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Toggle the specified GPIO pin.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the pin to be toggled. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_EnableHighSPeedLowVoltage

Function name

void HAL_GPIO_EnableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Enable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

HAL_GPIO_DisableHighSPeedLowVoltage

Function name

void HAL_GPIO_DisableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Disable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

HAL_GPIO_LockPin

Function name

HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Lock GPIO Pins configuration registers.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the port bits to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).

Return values

- **None:**

Notes

- The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.
- The configuration of the locked GPIO pins can no longer be modified until the next reset.

HAL_GPIO_EXTI_IRQHandler

Function name

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

Function description

Handle EXTI interrupt request.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Rising_Callback

Function name

void HAL_GPIO_EXTI_Rising_Callback (uint16_t GPIO_Pin)

Function description

EXTI line rising detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Falling_Callback

Function name

void HAL_GPIO_EXTI_Falling_Callback (uint16_t GPIO_Pin)

Function description

EXTI line falling detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_ConfigPinAttributes

Function name

void HAL_GPIO_ConfigPinAttributes (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, uint32_t PinAttributes)

Function description

Configure the GPIO pins attributes.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the pin(s) to configure the secure attribute
- **PinAttributes:** specifies the pin(s) to be set in secure mode, other being set non secured.

Return values

- **None:**

Notes

- Available attributes are to secure GPIO pin(s), so this function is only available in secure

HAL_GPIO_GetConfigPinAttributes

Function name

HAL_StatusTypeDef HAL_GPIO_GetConfigPinAttributes (const GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, uint32_t * pPinAttributes)

Function description

Get the GPIO pins attributes.

Parameters

- **GPIOx:** or LPGPIOx: where x can be (A..I) for the GPIO and (1) for LPGPIO to select the the corresponding peripheral for STM32U5 family
- **GPIO_Pin:** specifies the single pin to get the secure attribute from
- **pPinAttributes:** pointer to return the pin attributes.

Return values

- **HAL:** Status.

Notes

- Available attributes are to secure GPIO pin(s), so this function is only available in secure

31.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

31.3.1 GPIO

GPIO

GPIO attributes

GPIO_PIN_SEC

Secure pin attribute

GPIO_PIN_NSEC

Non-secure pin attribute

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_RISING_IT

Description:

- Check whether the specified EXTI line is rising edge asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_RISING_IT

Description:

- Clear the EXTI's line rising pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_FALLING_IT

Description:

- Check whether the specified EXTI line is falling edge asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FALLING_IT

Description:

- Clear the EXTI's line falling pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT

Description:

- Clear the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line(s).

Parameters:

- __EXTI_LINE__: specifies the EXTI line to set. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Check whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__: specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clear the EXTI line pending flags.

Parameters:

- __EXTI_LINE__: specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

GPIO mode

GPIO_MODE_INPUT

< Input Floating Mode Output Push Pull Mode

GPIO_MODE_OUTPUT_PP

Output Open Drain Mode

GPIO_MODE_OUTPUT_OD

Alternate Function Push Pull Mode

GPIO_MODE_AF_PP

Alternate Function Open Drain Mode

GPIO_MODE_AF_OD

Analog Mode

GPIO_MODE_ANALOG

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_RISING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

GPIO pins

GPIO_PIN_0

GPIO_PIN_1

GPIO_PIN_2

GPIO_PIN_3

GPIO_PIN_4

GPIO_PIN_5

GPIO_PIN_6

GPIO_PIN_7

GPIO_PIN_8

GPIO_PIN_9

GPIO_PIN_10

GPIO_PIN_11

GPIO_PIN_12

GPIO_PIN_13

GPIO_PIN_14

GPIO_PIN_15

GPIO_PIN_ALL

GPIO_PIN_MASK

GPIO pull

GPIO_NOPULL

No Pull-up or Pull-down activation

GPIO_PULLUP

Pull-up activation

GPIO_PULLDOWN

Pull-down activation

GPIO speed

GPIO_SPEED_FREQ_LOW

Low speed

GPIO_SPEED_FREQ_MEDIUM

Medium speed

GPIO_SPEED_FREQ_HIGH

High speed

GPIO_SPEED_FREQ_VERY_HIGH

Very-high speed

32 HAL GPIO Extension Driver

32.1 GPIOEx Firmware driver registers structures

32.1.1 LPGPIO_MapTypeDef

LPGPIO_MapTypeDef is defined in the stm32u5xx_hal_gpio_ex.h

Data Fields

- ***GPIO_TypeDef * GPIO_PORT***
- ***uint32_t Pin_Pos***

Field Documentation

- ***GPIO_TypeDef* LPGPIO_MapTypeDef::GPIO_PORT***
- ***uint32_t LPGPIO_MapTypeDef::Pin_Pos***

32.2 GPIOEx Firmware driver defines

The following section lists the various define and macros of the module.

32.2.1 GPIOEx

GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_RTC_50HZ

GPIO_AF0_MCO

GPIO_AF0_SWJ

GPIO_AF0_TRACE

GPIO_AF0_LPTIM1

GPIO_AF0_CSLEEP

GPIO_AF0_CSTOP

GPIO_AF0_CRS

GPIO_AF0_SRDSTOP

GPIO_AF1_TIM1

GPIO_AF1_TIM2

GPIO_AF1_TIM5

GPIO_AF1_TIM8

GPIO_AF1_LPTIM1

GPIO_AF1_IR

GPIO_AF2_TIM1

GPIO_AF2_TIM2

GPIO_AF2_TIM3

GPIO_AF2_TIM4

GPIO_AF2_TIM5

GPIO_AF2_LPTIM1

GPIO_AF2_LPTIM2

GPIO_AF2_LPTIM3

GPIO_AF2_I2C5

GPIO_AF2_I2C6

GPIO_AF2_GFXTIM

GPIO_AF3_I2C4

GPIO_AF3_OCTOSPI1

GPIO_AF3_SAI1

GPIO_AF3_SPI2

GPIO_AF3_TIM1

GPIO_AF3_TIM8

GPIO_AF3_TIM8_COMP1

GPIO_AF3_TIM8_COMP2

GPIO_AF3_TIM1_COMP1

GPIO_AF3_TIM1_COMP2

GPIO_AF3_USART2

GPIO_AF3_ADF1

GPIO_AF3_USB_HS

GPIO_AF4_I2C1

GPIO_AF4_I2C2

GPIO_AF4_I2C3

GPIO_AF4_I2C4

GPIO_AF4_PSSI

GPIO_AF4_DCMI

GPIO_AF4_LPTIM3

GPIO_AF4_I2C5

GPIO_AF5_I2C4

GPIO_AF5_OCTOSPI1

GPIO_AF5_OCTOSPI2

GPIO_AF5_SPI1

GPIO_AF5_SPI2

GPIO_AF5_SPI3

GPIO_AF5_DCMI

GPIO_AF5_MDF1

GPIO_AF5_PSSI

GPIO_AF5_GFXTIM

GPIO_AF6_OCTOSPI1

GPIO_AF6_OCTOSPI2

GPIO_AF6_MDF1

GPIO_AF6_SPI3

GPIO_AF6_I2C3

GPIO_AF7_USART1

GPIO_AF7_USART2

GPIO_AF7_USART3

GPIO_AF7_USART6

GPIO_AF7_LTDC

GPIO_AF8_LPUART1

GPIO_AF8_UART4

GPIO_AF8_UART5

GPIO_AF8_SDMMC1

GPIO_AF8_SDMMC2

GPIO_AF8_LTDC

GPIO_AF8_HSPI1

GPIO_AF9_FDCAN1

GPIO_AF9_TSC

GPIO_AF10_DCMI

GPIO_AF10_PSSI

GPIO_AF10_USB

GPIO_AF10_OCTOSPI1

GPIO_AF10_OCTOSPI2

GPIO_AF10_CRS

GPIO_AF10_USB_HS

GPIO_AF10_DSI

GPIO_AF10_GFXTIM

GPIO_AF11_UCPD1

GPIO_AF11_SDMMC2

GPIO_AF11_LPGPIO1

GPIO_AF11_FMC

GPIO_AF11_DSI

GPIO_AF11_GFXTIM

GPIO_AF12_COMP1

GPIO_AF12_COMP2

GPIO_AF12_FMC

GPIO_AF12_TIM1_COMP1

GPIO_AF12_TIM1_COMP2

GPIO_AF12_TIM8_COMP2

GPIO_AF12_SDMMC1

GPIO_AF12_SDMMC2

GPIO_AF13_SAI1

GPIO_AF13_SAI2

GPIO_AF13_LPTIM4

GPIO_AF13_LPTIM2

GPIO_AF13_GFXTIM

GPIO_AF14_LPTIM2

GPIO_AF14_LPTIM3

GPIO_AF14_TIM2

GPIO_AF14_TIM15

GPIO_AF14_TIM15_COMP1

GPIO_AF14_TIM16

GPIO_AF14_TIM16_COMP1

GPIO_AF14_TIM17

GPIO_AF14_TIM17_COMP1

GPIO_AF14_FMC

GPIO_AF15_EVENTOUT

IS_GPIO_AF

GPIOEx Get Port Index

GPIO_GET_INDEX

33 HAL GPU2D Generic Driver

33.1 GPU2D Firmware driver registers structures

33.1.1 __GPU2D_HandleTypeDef

__GPU2D_HandleTypeDef is defined in the stm32u5xx_hal_gpu2d.h

Data Fields

- ***GPU2D_TypeDef Instance***
- ***HAL_LockTypeDef Lock***
- ***HAL_GPU2D_StateTypeDef State***
- ***uint32_t ErrorCode***
- ***void(* CommandListCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***GPU2D_TypeDef __GPU2D_HandleTypeDef::Instance***
  GPU2D register base address.
- ***HAL_LockTypeDef __GPU2D_HandleTypeDef::Lock***
  GPU2D lock.
- ***HAL_GPU2D_StateTypeDef __GPU2D_HandleTypeDef::State***
  GPU2D transfer state.
- ***uint32_t __GPU2D_HandleTypeDef::ErrorCode***
  GPU2D error code.
- ***void(* __GPU2D_HandleTypeDef::CommandListCpltCallback)(struct __GPU2D_HandleTypeDef *hgpu2d, uint32_t CmdListID)***
  GPU2D Command Complete Callback
- ***void(* __GPU2D_HandleTypeDef::MspInitCallback)(struct __GPU2D_HandleTypeDef *hgpu2d)***
  GPU2D Msp Init callback
- ***void(* __GPU2D_HandleTypeDef::MspDeInitCallback)(struct __GPU2D_HandleTypeDef *hgpu2d)***
  GPU2D Msp DeInit callback

33.2 GPU2D Firmware driver API description

The following section lists the various functions of the GPU2D library.

33.2.1 How to use this driver

1. Peripheral control is exclusively done by the accompanying middleware library.

Interrupt mode IO operation

1. Configure the GPU2D hardware to perform graphics operation using the Third Party MW Library APIs.
2. Submit command List to the hardware.
3. Wait indefinitely for the completion of submitted Command List by GPU2D hardware.
4. Use HAL_GPU2D_IRQHandler() called under GPU2D_IRQHandler() interrupt subroutine.
5. At the end of Command List execution HAL_GPU2D_IRQHandler() function is executed and user can add his own function by customization of function pointer
6. CommandListCpltCallback (member of GPU2D handle structure) to notify the upper level about the completion of Command List execution.

7. Callback HAL_GPU2D_CommandListCpltCallback is invoked when the GPU2D hardware executes the programmed command list (Command List execution completion).
   – This callback is called when the compilation defines USE_HAL_GPU2D_REGISTER_CALLBACKS is set to 0 or not defined.
   – This callback should be implemented in the application side. It should notify the upper level that the programmed command list is completed.
8. To control the GPU2D state, use the following function: HAL_GPU2D_GetState().
9. To read the GPU2D error code, use the following function: HAL_GPU2D_GetError().

GPU2D HAL driver macros list

Below the list of most used macros in GPU2D HAL driver :

- __HAL_GPU2D_RESET_HANDLE_STATE: Reset GPU2D handle state.
- __HAL_GPU2D_GET_FLAG: Get the GPU2D pending flags.
- __HAL_GPU2D_CLEAR_FLAG: Clear the GPU2D pending flags.
- __HAL_GPU2D_GET_IT_SOURCE: Check whether the specified GPU2D interrupt is enabled or not.

Callback registration

1. The compilation define USE_HAL_GPU2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL_GPU2D_RegisterCallback() to register a user callback.
2. Function @ref HAL_GPU2D_RegisterCallback() allows to register following callbacks: (+) CommandListCpltCallback : callback for Command List completion. (+) MspInitCallback : GPU2D MspInit. (+) MspDeInitCallback : GPU2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function @ref HAL_GPU2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_GPU2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) CommandListCpltCallback : callback for Command List completion. (+) MspInitCallback : GPU2D MspInit. (+) MspDeInitCallback : GPU2D MspDeInit. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_GPU2D_RegisterCallback before calling @ref HAL_GPU2D_DeInit or @ref HAL_GPU2D_Init function. When The compilation define USE_HAL_GPU2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

(#) The compilation define USE_HAL_GPU2D_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function @ref HAL_GPU2D_RegisterCallback() to register a user callback. (#) Function @ref HAL_GPU2D_RegisterCallback() allows to register following callbacks:

- CommandListCpltCallback : callback for Command List completion.
- MspInitCallback : GPU2D MspInit.
- MspDeInitCallback : GPU2D MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function @ref HAL_GPU2D_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. @ref HAL_GPU2D_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- CommandListCpltCallback : callback for Command List completion.
- MspInitCallback : GPU2D MspInit.
- MspDeInitCallback : GPU2D MspDeInit. Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using @ref HAL_GPU2D_RegisterCallback before calling @ref HAL_GPU2D_DeInit or @ref HAL_GPU2D_Init function. When The compilation define USE_HAL_GPU2D_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

Note: You can refer to the GPU2D HAL driver header file for more useful macros

33.2.2 Initialization and Configuration functions

This section provides functions allowing to:
- Initialize and configure the GPU2D
- De-initialize the GPU2D

This section contains the following APIs:
- *HAL_GPU2D_Init()*
- *HAL_GPU2D_DeInit()*
- *HAL_GPU2D_MspInit()*
- *HAL_GPU2D_MspDeInit()*
- *HAL_GPU2D_RegisterCallback()*
- *HAL_GPU2D_UnRegisterCallback()*
- *HAL_GPU2D_RegisterCommandListCpltCallback()*
- *HAL_GPU2D_UnRegisterCommandListCpltCallback()*

33.2.3 IO operation functions

This section provides functions allowing to:
- Read GPU2D Register value.
- Write a value to GPU2D Register.
- handle GPU2D interrupt request.
- Command List Complete Transfer Complete callback.

This section contains the following APIs:
- *HAL_GPU2D_ReadRegister()*
- *HAL_GPU2D_WriteRegister()*
- *HAL_GPU2D_IRQHandler()*
- *HAL_GPU2D_ER_IRQHandler()*
- *HAL_GPU2D_CommandListCpltCallback()*
- *HAL_GPU2D_ErrorCallback()*

33.2.4 Peripheral State and Errors functions

This subsection provides functions allowing to:
- Get the GPU2D state
- Get the GPU2D error code

This section contains the following APIs:
- *HAL_GPU2D_GetState()*
- *HAL_GPU2D_GetError()*

33.2.5 Detailed description of functions

HAL_GPU2D_Init

Function name

HAL_StatusTypeDef HAL_GPU2D_Init (GPU2D_HandleTypeDef * hgpu2d)

Function description

Initialize the GPU2D according to the specified parameters in the GPU2D_InitTypeDef and create the associated handle.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **HAL:** status

HAL_GPU2D_DeInit

Function name

HAL_StatusTypeDef HAL_GPU2D_DeInit (GPU2D_HandleTypeDef * hgpu2d)

Function description

Deinitializes the GPU2D peripheral registers to their default reset values.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **None:**

HAL_GPU2D_MspInit

Function name

void HAL_GPU2D_MspInit (GPU2D_HandleTypeDef * hgpu2d)

Function description

Initializes the GPU2D MSP.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **None:**

HAL_GPU2D_MspDeInit

Function name

void HAL_GPU2D_MspDeInit (GPU2D_HandleTypeDef * hgpu2d)

Function description

DeInitializes the GPU2D MSP.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **None:**

HAL_GPU2D_RegisterCallback

Function name

HAL_StatusTypeDef HAL_GPU2D_RegisterCallback (GPU2D_HandleTypeDef * hgpu2d, HAL_GPU2D_CallbackIDTypeDef CallbackID, pGPU2D_CallbackTypeDef pCallback)

Function description

Register a User GPU2D callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hgpu2d:** GPU2D handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_GPU2D_MSPINIT_CB_ID GPU2D MspInit callback ID
  – HAL_GPU2D_MSPDEINIT_CB_ID GPU2D MspDeInit callback ID
- **pCallback:** pointer to the callback function

Return values

- **HAL:** status

Notes

- Weak predefined callback is defined for HAL_GPU2D_MSPINIT_CB_ID and HAL_GPU2D_MSPDEINIT_CB_ID

HAL_GPU2D_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_GPU2D_UnRegisterCallback (GPU2D_HandleTypeDef * hgpu2d, HAL_GPU2D_CallbackIDTypeDef CallbackID)

Function description

Unregister a GPU2D callback GPU2D Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hgpu2d:** GPU2D handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_GPU2D_MSPINIT_CB_ID GPU2D MspInit callback ID
  – HAL_GPU2D_MSPDEINIT_CB_ID GPU2D MspDeInit callback ID

Return values

- **status:**

Notes

- Callback pointers will be set to legacy weak predefined callbacks for HAL_GPU2D_MSPINIT_CB_ID and HAL_GPU2D_MSPDEINIT_CB_ID

HAL_GPU2D_RegisterCommandListCpltCallback

Function name

HAL_StatusTypeDef HAL_GPU2D_RegisterCommandListCpltCallback (GPU2D_HandleTypeDef * hgpu2d, pGPU2D_CommandListCpltCallbackTypeDef pCallback)

Function description

Register GPU2D Command List Complete Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hgpu2d:** GPU2D handle
- **pCallback:** pointer to the Command List Complete Callback function

Return values

- **status:**

Notes

- Weak predefined callback is defined for Command List Complete

HAL_GPU2D_UnRegisterCommandListCpltCallback

Function name

HAL_StatusTypeDef HAL_GPU2D_UnRegisterCommandListCpltCallback (GPU2D_HandleTypeDef * hgpu2d)

Function description

Unregister a GPU2D Command List Complete Callback GPU2D Command List Complete Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hgpu2d:** GPU2D handle

Return values

- **status:**

Notes

- Callback pointer will be invalidate (NULL value)

HAL_GPU2D_ReadRegister

Function name

uint32_t HAL_GPU2D_ReadRegister (GPU2D_HandleTypeDef * hgpu2d, uint32_t offset)

Function description

Read GPU2D Register.

Parameters

- **hgpu2d:** Pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.
- **offset:** The register offset from GPU2D base address to read.

Return values

- **Register:** value

HAL_GPU2D_WriteRegister

Function name

HAL_StatusTypeDef HAL_GPU2D_WriteRegister (GPU2D_HandleTypeDef * hgpu2d, uint32_t offset, uint32_t value)

Function description

Write a value to GPU2D Register.

Parameters

- **hgpu2d:** Pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.
- **offset:** The register offset from GPU2D base address to write.
- **value:** The value to be written to provided register.

Return values

- **HAL:** status

HAL_GPU2D_IRQHandler

Function name

void HAL_GPU2D_IRQHandler (GPU2D_HandleTypeDef * hgpu2d)

Function description

Handle GPU2D interrupt request.

Parameters

- **hgpu2d:** Pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **None:**

HAL_GPU2D_ER_IRQHandler

Function name

void HAL_GPU2D_ER_IRQHandler (GPU2D_HandleTypeDef * hgpu2d)

Function description

Handle GPU2D Error interrupt request.

Parameters

- **hgpu2d:** Pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **None:**

HAL_GPU2D_CommandListCpltCallback

Function name

void HAL_GPU2D_CommandListCpltCallback (GPU2D_HandleTypeDef * hgpu2d, uint32_t CmdListID)

Function description

Command List Complete callback.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.
- **CmdListID:** Command list ID that got completed.

Return values

- **None:**

HAL_GPU2D_ErrorCallback

Function name

void HAL_GPU2D_ErrorCallback (GPU2D_HandleTypeDef * hgpu2d)

Function description

Error handler callback.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **None:**

HAL_GPU2D_GetState

Function name

HAL_GPU2D_StateTypeDef HAL_GPU2D_GetState (GPU2D_HandleTypeDef const *const hgpu2d)

Function description

Return the GPU2D state.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for the GPU2D.

Return values

- **GPU2D:** state

HAL_GPU2D_GetError

Function name

uint32_t HAL_GPU2D_GetError (GPU2D_HandleTypeDef const *const hgpu2d)

Function description

Return the GPU2D error code.

Parameters

- **hgpu2d:** pointer to a GPU2D_HandleTypeDef structure that contains the configuration information for GPU2D.

Return values

- **GPU2D:** Error Code

33.3 GPU2D Firmware driver defines

The following section lists the various define and macros of the module.

33.3.1 GPU2D

GPU2D

GPU2D Error Code definition

HAL_GPU2D_ERROR_NONE

No error

HAL_GPU2D_ERROR_TIMEOUT

Timeout error

HAL_GPU2D_ERROR_INVALID_CALLBACK

Invalid callback error

GPU2D Exported Macros

__HAL_GPU2D_RESET_HANDLE_STATE

Description:

- Reset GPU2D handle state.

Parameters:

- __HANDLE__: specifies the GPU2D handle.

Return value:

- None

__HAL_GPU2D_GET_FLAG

Description:
- Get the GPU2D pending flags.

Parameters:
- __HANDLE__: GPU2D handle
- __FLAG__: flag to check. This parameter can be one of the following values:
  - GPU2D_FLAG_CLC: Command List Complete interrupt mask

Return value:
- The: state of FLAG.

__HAL_GPU2D_CLEAR_FLAG

Description:
- Clear the GPU2D pending flags.

Parameters:
- __HANDLE__: GPU2D handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - GPU2D_FLAG_CLC: Command List Complete interrupt mask

Return value:
- None

__HAL_GPU2D_GET_IT_SOURCE

Description:
- Check whether the specified GPU2D interrupt source is enabled or not.

Parameters:
- __HANDLE__: GPU2D handle
- __INTERRUPT__: specifies the GPU2D interrupt source to check. This parameter can be one of the following values:
  - GPU2D_IT_CLC: Command List Complete interrupt mask

Return value:
- The: state of INTERRUPT source.

GPU2D Flag definition

GPU2D_FLAG_CLC

Command List Complete Interrupt Flag

GPU2D Interrupt configuration definition

GPU2D_IT_CLC

Command List Complete Interrupt

GPU2D Last Register Offset

GPU2D_OFFSET

Last GPU2D Register Offset

Common write and read registers Macros

GPU2D_WriteReg

Description:
- Write a value in GPU2D register.

Parameters:
- __INSTANCE__: GPU2D Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

GPU2D_ReadReg

Description:
- Read a value in GPU2D register.

Parameters:
- __INSTANCE__: GPU2D Instance
- __REG__: Register to be read

Return value:
- Register: value

34 HAL GTZC Generic Driver

34.1 GTZC Firmware driver registers structures

34.1.1 MPCBB_Attribute_ConfigTypeDef

MPCBB_Attribute_ConfigTypeDef is defined in the stm32u5xx_hal_gtzc.h

Data Fields

- ***uint32_t MPCBB_SecConfig_array***
- ***uint32_t MPCBB_PrivConfig_array***
- ***uint32_t MPCBB_LockConfig_array***

Field Documentation

- ***uint32_t MPCBB_Attribute_ConfigTypeDef::MPCBB_SecConfig_array[GTZC_MPCBB_NB_VCTR_REG_MAX]***
  Each element specifies secure access mode for a super-block. Each bit corresponds to a block inside the super-block. 0 means non-secure, 1 means secure
- ***uint32_t MPCBB_Attribute_ConfigTypeDef::MPCBB_PrivConfig_array[GTZC_MPCBB_NB_VCTR_REG_MAX]***
  Each element specifies privilege access mode for a super-block. Each bit corresponds to a block inside the super-block. 0 means non-privilege, 1 means privilege
- ***uint32_t MPCBB_Attribute_ConfigTypeDef::MPCBB_LockConfig_array[GTZC_MPCBB_NB_LCK_VCTR_REG_MAX]***
  Each bit specifies the lock configuration of a super-block (32 blocks). 0 means unlocked, 1 means locked

34.1.2 MPCBB_ConfigTypeDef

MPCBB_ConfigTypeDef is defined in the stm32u5xx_hal_gtzc.h

Data Fields

- ***uint32_t SecureRWIllegalMode***
- ***uint32_t InvertSecureState***
- ***MPCBB_Attribute_ConfigTypeDef AttributeConfig***

Field Documentation

- ***uint32_t MPCBB_ConfigTypeDef::SecureRWIllegalMode***
  Secure read/write illegal access field. It can be a value of ***GTZC_MPCBB_SecureRWIllegalMode***
- ***uint32_t MPCBB_ConfigTypeDef::InvertSecureState***
  Default security state field (can be inverted or not). It can be a value of ***GTZC_MPCBB_InvertSecureState***
- ***MPCBB_Attribute_ConfigTypeDef MPCBB_ConfigTypeDef::AttributeConfig***
  MPCBB attribute configuration sub-structure

34.1.3 MPCWM_ConfigTypeDef

MPCWM_ConfigTypeDef is defined in the stm32u5xx_hal_gtzc.h

Data Fields

- ***uint32_t AreaId***
- ***uint32_t Offset***
- ***uint32_t Length***
- ***uint32_t Attribute***
- ***uint32_t Lock***
- ***uint32_t AreaStatus***

Field Documentation

- ***uint32_t MPCWM_ConfigTypeDef::AreaId***
  Area identifier field. It can be a value of ***GTZC_MPCWM_AreaId***

- ***uint32_t MPCWM_ConfigTypeDef::Offset***
  Offset of the watermark area, starting from the selected memory base address. It must aligned on 128KB for FMC and OCTOSPI memories, and on 32-byte for BKPSRAM
- ***uint32_t MPCWM_ConfigTypeDef::Length***
  Length of the watermark area, starting from the selected Offset. It must aligned on 128KB for FMC and OCTOSPI memories, and on 32-byte for BKPSRAM
- ***uint32_t MPCWM_ConfigTypeDef::Attribute***
  Attributes of the watermark area. It can be a value of ***GTZC_MPCWM_Attribute***
- ***uint32_t MPCWM_ConfigTypeDef::Lock***
  Lock of the watermark area. It can be a value of ***GTZC_MPCWM_Lock***
- ***uint32_t MPCWM_ConfigTypeDef::AreaStatus***
  Status of the watermark area. It can be set to ENABLE or DISABLE

34.2 GTZC Firmware driver API description

The following section lists the various functions of the GTZC library.

34.2.1 GTZC main features

- Global TrustZone Controller (GTZC) composed of three sub-blocks:
  - TZSC: TrustZone security controller This sub-block defines the secure/privileged state of master and slave peripherals. It also controls the secure/privileged state of subregions for the watermark memory peripheral controller (MPCWM).
  - MPCBB: Block-Based memory protection controller This sub-block defines the secure/privileged state of all blocks (512-byte pages) of the associated SRAM.
  - TZIC: TrustZone illegal access controller This sub-block gathers all illegal access events in the system and generates a secure interrupt towards NVIC.
- These sub-blocks are used to configure TrustZone system security in a product having bus agents with programmable-security and privileged attributes (securable) such as:
  - on-chip RAM with programmable secure and/or privilege blocks (pages)
  - AHB and APB peripherals with programmable security and/or privilege access
  - AHB master granted as secure and/or privilege
  - off-chip memories with secure and/or privilege areas
- TZIC accessible only with secure privileged transactions.
- Secure and non-secure access supported for privileged and unprivileged part of TZSC and MPCBB
- Set of registers to define product security settings:
  - Secure and privilege blocks for internal memories
  - Secure and privilege regions for external memories
  - Secure and privileged access mode for securable peripherals
    ==

34.2.2 How to use this driver

The GTZC HAL driver can be used as follows:

1. Configure or get back securable peripherals attributes using HAL_GTZC_TZSC_ConfigPeriphAttributes() / HAL_GTZC_TZSC_GetConfigPeriphAttributes()
2. Configure or get back MPCWM memories attributes using HAL_GTZC_TZSC_MPCWM_ConfigMemAttributes() / HAL_GTZC_TZSC_MPCWM_GetConfigMemAttributes()
3. Lock TZSC sub-block or get lock status using HAL_GTZC_TZSC_Lock() / HAL_GTZC_TZSC_GetLock()
4. Configure or get back MPCBB memories complete configuration using HAL_GTZC_MPCBB_ConfigMem() / HAL_GTZC_MPCBB_GetConfigMem()
5. Configure or get back MPCBB memories attributes using HAL_GTZC_MPCBB_ConfigMemAttributes() / HAL_GTZC_MPCBB_GetConfigMemAttributes()

6. Lock MPCBB configuration or get lock status using HAL_GTZC_MPCBB_Lock() / HAL_GTZC_MPCBB_GetLock()
7. Lock MPCBB super-blocks or get lock status using HAL_GTZC_MPCBB_LockConfig() / HAL_GTZC_MPCBB_GetLockConfig()
8. Illegal access detection can be configured through TZIC sub-block using following functions: HAL_GTZC_TZIC_DisableIT() / HAL_GTZC_TZIC_EnableIT()
9. Illegal access flags can be retrieved through HAL_GTZC_TZIC_GetFlag() and HAL_GTZC_TZIC_ClearFlag() functions
10. Illegal access interrupt service routines are served by HAL_GTZC_IRQHandler() and user can add his own code using HAL_GTZC_TZIC_Callback()

34.2.3 Detailed description of functions

HAL_GTZC_TZSC_ConfigPeriphAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_ConfigPeriphAttributes (uint32_t PeriphId, uint32_t PeriphAttributes)

Function description

Configure TZSC on a single peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.
- **PeriphAttributes:** Peripheral attributes, see GTZC TZSC peripheral attribute values.

Return values

- **HAL:** status.

Notes

- Secure and non-secure attributes can only be set from the secure state when the system implements the security (TZEN=1).
- Privilege and non-privilege attributes can only be set from the privilege state when TZEN=0 or TZEN=1
- Security and privilege attributes can be set independently.
- Default state is non-secure and unprivileged access allowed.

HAL_GTZC_TZSC_GetConfigPeriphAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_GetConfigPeriphAttributes (uint32_t PeriphId, uint32_t * PeriphAttributes)

Function description

Get TZSC configuration on a single peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.
- **PeriphAttributes:** Peripheral attribute pointer. This parameter can be a value of GTZC TZSC peripheral attribute values. If PeriphId target a single peripheral, pointer on a single element. If all peripherals selected (GTZC_PERIPH_ALL), pointer to an array of GTZC_TZSC_PERIPH_NUMBER elements is to be provided.

Return values

- **HAL:** status.

HAL_GTZC_TZSC_MPCWM_ConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_MPCWM_ConfigMemAttributes (uint32_t MemBaseAddress, const MPCWM_ConfigTypeDef * pMPCWM_Desc)

Function description

Configure a TZSC-MPCWM area.

Parameters

- **MemBaseAddress:** WM identifier.
- **pMPCWM_Desc:** TZSC-MPCWM descriptor pointer. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_TZSC_MPCWM_GetConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_TZSC_MPCWM_GetConfigMemAttributes (uint32_t MemBaseAddress, MPCWM_ConfigTypeDef * pMPCWM_Desc)

Function description

Get a TZSC-MPCWM area configuration.

Parameters

- **MemBaseAddress:** WM identifier.
- **pMPCWM_Desc:** pointer to a TZSC-MPCWM descriptor. When the WaterMark memory supports two sub-regions A and B. pMPCWM_Desc argument must point to an array of two MPCWM_ConfigTypeDef structures. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_TZSC_Lock

Function name

void HAL_GTZC_TZSC_Lock (GTZC_TZSC_TypeDef * TZSC_Instance)

Function description

Lock TZSC configuration.

Parameters

- **TZSC_Instance:** TZSC sub-block instance.

Notes

- This function locks the configuration of TZSC_SECCFGRx and TZSC_PRIVCFGRx registers until next reset

HAL_GTZC_TZSC_GetLock

Function name

uint32_t HAL_GTZC_TZSC_GetLock (const GTZC_TZSC_TypeDef * TZSC_Instance)

Function description

Get TZSC configuration lock state.

Parameters

- **TZSC_Instance:** TZSC sub-block instance.

Return values

- **Lock:** State (GTZC_TZSC_LOCK_OFF or GTZC_TZSC_LOCK_ON)

HAL_GTZC_MPCBB_ConfigMem

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_ConfigMem (uint32_t MemBaseAddress, const MPCBB_ConfigTypeDef * pMPCBB_desc)

Function description

Set a complete MPCBB configuration on the SRAM passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.
- **pMPCBB_desc:** pointer to MPCBB descriptor. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_GetConfigMem

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetConfigMem (uint32_t MemBaseAddress, MPCBB_ConfigTypeDef * pMPCBB_desc)

Function description

Get a complete MPCBB configuration on the SRAM passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.
- **pMPCBB_desc:** pointer to a MPCBB descriptor. The structure description is available in GTZC Exported Types.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_ConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_ConfigMemAttributes (uint32_t MemAddress, uint32_t NbBlocks, const uint32_t * pMemAttributes)

Function description

Set a MPCBB attribute configuration on the SRAM passed as parameter for a number of blocks.

Parameters

- **MemAddress:** MPCBB identifier, and start block to configure (must be 512 Bytes aligned).
- **NbBlocks:** Number of blocks to configure (Block size is 512 Bytes).
- **pMemAttributes:** pointer to an array (containing "NbBlocks" elements), with each element must be GTZC_MPCBB_BLOCK_NSEC or GTZC_MPCBB_BLOCK_SEC, and GTZC_MPCBB_BLOCK_NPRIV or GTZC_MPCBB_BLOCK_PRIV.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_GetConfigMemAttributes

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetConfigMemAttributes (uint32_t MemAddress, uint32_t NbBlocks, uint32_t * pMemAttributes)

Function description

Get a MPCBB attribute configuration on the SRAM passed as parameter for a number of blocks.

Parameters

- **MemAddress:** MPCBB identifier, and start block to get configuration (must be 512 Bytes aligned).
- **NbBlocks:** Number of blocks to get configuration.
- **pMemAttributes:** pointer to an array (containing "NbBlocks" elements), with each element will be GTZC_MPCBB_BLOCK_NSEC or GTZC_MPCBB_BLOCK_SEC, and GTZC_MPCBB_BLOCK_NPRIV or GTZC_MPCBB_BLOCK_PRIV.

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_LockConfig

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_LockConfig (uint32_t MemAddress, uint32_t NbSuperBlocks, const uint32_t * pLockAttributes)

Function description

Lock MPCBB super-blocks on the SRAM passed as parameter.

Parameters

- **MemAddress:** MPCBB start-address of super-block to configure (must be 16KBytes aligned).
- **NbSuperBlocks:** Number of super-blocks to configure.
- **pLockAttributes:** pointer to an array (containing "NbSuperBlocks" elements), with for each element: value 0 super-block is unlocked, value 1 super-block is locked (corresponds to GTZC_MPCBB_SUPERBLOCK_UNLOCKED and GTZC_MPCBB_SUPERBLOCK_LOCKED values).

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_GetLockConfig

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetLockConfig (uint32_t MemAddress, uint32_t NbSuperBlocks, uint32_t * pLockAttributes)

Function description

Get MPCBB super-blocks lock configuration on the SRAM passed as parameter.

Parameters

- **MemAddress:** MPCBB start-address of super-block to get configuration (must be 16KBytes aligned).
- **NbSuperBlocks:** Number of super-blocks to get configuration.
- **pLockAttributes:** pointer to an array (containing "NbSuperBlocks" elements), with for each element: value 0 super-block is unlocked, value 1 super-block is locked (corresponds to GTZC_MPCBB_SUPERBLOCK_UNLOCKED and GTZC_MPCBB_SUPERBLOCK_LOCKED values).

Return values

- **HAL:** status.

HAL_GTZC_MPCBB_Lock

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_Lock (uint32_t MemBaseAddress)

Function description

Lock a MPCBB configuration on the SRAM base address passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.

Return values

- **HAL:** status.

Notes

- This functions locks the control register of the MPCBB until next reset.

HAL_GTZC_MPCBB_GetLock

Function name

HAL_StatusTypeDef HAL_GTZC_MPCBB_GetLock (uint32_t MemBaseAddress, uint32_t * pLockState)

Function description

Get MPCBB configuration lock state on the SRAM base address passed as parameter.

Parameters

- **MemBaseAddress:** MPCBB identifier.
- **pLockState:** pointer to Lock State (GTZC_MPCBB_LOCK_OFF or GTZC_MPCBB_LOCK_ON).

Return values

- **HAL:** status.

HAL_GTZC_TZIC_DisableIT

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_DisableIT (uint32_t PeriphId)

Function description

Disable the interrupt associated to a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.

Return values

- **HAL:** status.

HAL_GTZC_TZIC_EnableIT

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_EnableIT (uint32_t PeriphId)

Function description

Enable the interrupt associated to a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.

Return values

- **HAL:** status.

HAL_GTZC_TZIC_GetFlag

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_GetFlag (uint32_t PeriphId, uint32_t * pFlag)

Function description

Get TZIC flag on a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.
- **pFlag:** Pointer to the flags. If PeriphId target a single peripheral, pointer on a single element. If all peripherals selected (GTZC_PERIPH_ALL), pointer to an array of GTZC_TZIC_PERIPH_NUMBER elements. Element content is either GTZC_TZIC_NO_ILA_EVENT or GTZC_TZSC_ILA_EVENT_PENDING.

Return values

- **HAL:** status

HAL_GTZC_TZIC_ClearFlag

Function name

HAL_StatusTypeDef HAL_GTZC_TZIC_ClearFlag (uint32_t PeriphId)

Function description

Clear TZIC flag on a single TZIC peripheral or on all peripherals.

Parameters

- **PeriphId:** Peripheral identifier. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values. Use GTZC_PERIPH_ALL to select all peripherals.

Return values

- **HAL:** status.

HAL_GTZC_IRQHandler

Function name

void HAL_GTZC_IRQHandler (void)

Function description

This function handles GTZC TZIC interrupt request.

Return values

- **None.:**

HAL_GTZC_TZIC_Callback

Function name

void HAL_GTZC_TZIC_Callback (uint32_t PeriphId)

Function description

GTZC TZIC sub-block interrupt callback.

Parameters

- **PeriphId:** Peripheral identifier triggering the illegal access. This parameter can be a value of GTZC TZSC and TZIC Peripheral identifier values

Return values

- **None.:**

34.3 GTZC Firmware driver defines

The following section lists the various define and macros of the module.

34.3.1 GTZC

GTZC

GTZC Exported Macros

HAL_GTZC_TZSC_GET_ARRAY_INDEX

HAL_GTZC_TZIC_GET_ARRAY_INDEX

HAL_GTZC_TZSC_GET_INSTANCE

HAL_GTZC_TZIC_GET_INSTANCE

GTZC Exported Types

GTZC_MPCBB_NB_VCTR_REG_MAX

< Values needed for

GTZC_MPCBB_NB_LCK_VCTR_REG_MAX

GTZC MPCBB values

GTZC_MPCBB_BLOCK_SIZE

GTZC_MPCBB_SUPERBLOCK_SIZE

GTZC_MPCBB_SUPERBLOCK_UNLOCKED

GTZC_MPCBB_SUPERBLOCK_LOCKED

GTZC_MPCBB_BLOCK_NSEC

GTZC_MPCBB_BLOCK_SEC

GTZC_MPCBB_BLOCK_NPRIV

GTZC_MPCBB_BLOCK_PRIV

GTZC_MPCBB_LOCK_OFF

GTZC_MPCBB_LOCK_ON

GTZC MPCBB INVSECSTATE values

GTZC_MPCBB_INVSECSTATE_NOT_INVERTED

GTZC_MPCBB_INVSECSTATE_INVERTED

GTZC MPCBB SRWILADIS values

GTZC_MPCBB_SRWILADIS_ENABLE

GTZC_MPCBB_SRWILADIS_DISABLE

GTZC MPCWM area identifier values

GTZC_TZSC_MPCWM_ID1

GTZC_TZSC_MPCWM_ID2

GTZC MPCWM Attribute values

GTZC_TZSC_MPCWM_REGION_NSEC

GTZC_TZSC_MPCWM_REGION_SEC

GTZC_TZSC_MPCWM_REGION_NPRIV

GTZC_TZSC_MPCWM_REGION_PRIV

GTZC MPCWM values

GTZC_TZSC_MPCWM_GRANULARITY_1

GTZC_TZSC_MPCWM_GRANULARITY_2

GTZC MPCWM Lock values

GTZC_TZSC_MPCWM_LOCK_OFF

GTZC_TZSC_MPCWM_LOCK_ON

GTZC TZIC flag values

GTZC_TZIC_NO_ILA_EVENT

GTZC_TZIC_ILA_EVENT_PENDING

GTZC TZSC lock values

GTZC_TZSC_LOCK_OFF

GTZC_TZSC_LOCK_ON

GTZC TZSC peripheral attribute values

GTZC_TZSC_PERIPH_SEC

Secure attribute

GTZC_TZSC_PERIPH_NSEC

Non-secure attribute

GTZC_TZSC_PERIPH_PRIV

Privilege attribute

GTZC_TZSC_PERIPH_NPRIV

Non-privilege attribute

GTZC TZSC and TZIC Peripheral identifier values

GTZC_PERIPH_TIM2

GTZC_PERIPH_TIM3

GTZC_PERIPH_TIM4

GTZC_PERIPH_TIM5

GTZC_PERIPH_TIM6

GTZC_PERIPH_TIM7

GTZC_PERIPH_WWDG

GTZC_PERIPH_IWDG

GTZC_PERIPH_SPI2

GTZC_PERIPH_USART2

GTZC_PERIPH_USART3

GTZC_PERIPH_UART4

GTZC_PERIPH_UART5

GTZC_PERIPH_I2C1

GTZC_PERIPH_I2C2

GTZC_PERIPH_CRS

GTZC_PERIPH_I2C4

GTZC_PERIPH_LPTIM2

GTZC_PERIPH_FDCAN1

GTZC_PERIPH_UCPD1

GTZC_PERIPH_USART6

GTZC_PERIPH_I2C5

GTZC_PERIPH_I2C6

GTZC_PERIPH_TIM1

GTZC_PERIPH_SPI1

GTZC_PERIPH_TIM8

GTZC_PERIPH_USART1

GTZC_PERIPH_TIM15

GTZC_PERIPH_TIM16

GTZC_PERIPH_TIM17

GTZC_PERIPH_SAI1

GTZC_PERIPH_SAI2

GTZC_PERIPH_LTDCUSB

GTZC_PERIPH_DSI

GTZC_PERIPH_GFXTIM

GTZC_PERIPH_MDF1

GTZC_PERIPH_CORDIC

GTZC_PERIPH_FMAC

GTZC_PERIPH_CRC

GTZC_PERIPH_TSC

GTZC_PERIPH_DMA2D

GTZC_PERIPH_ICACHE_REG

GTZC_PERIPH_DCACHE1_REG

GTZC_PERIPH_ADC12

GTZC_PERIPH_DCMI_PSSI

GTZC_PERIPH_OTG

GTZC_PERIPH_AES

GTZC_PERIPH_HASH

GTZC_PERIPH_RNG

GTZC_PERIPH_PKA

GTZC_PERIPH_SAES

GTZC_PERIPH_OCTOSPIM

GTZC_PERIPH_SDMMC1

GTZC_PERIPH_SDMMC2

GTZC_PERIPH_FSMC_REG

GTZC_PERIPH_OCTOSPI1_REG

GTZC_PERIPH_OCTOSPI2_REG

GTZC_PERIPH_RAMCFG

GTZC_PERIPH_GPU2D

GTZC_PERIPH_GFXMMU

GTZC_PERIPH_GFXMMU_REG

GTZC_PERIPH_HSPI1_REG

GTZC_PERIPH_DCACHE2_REG

GTZC_PERIPH_JPEG

GTZC_PERIPH_GPDMA1

GTZC_PERIPH_FLASH_REG

GTZC_PERIPH_FLASH

GTZC_PERIPH_OTFDEC2

GTZC_PERIPH_OTFDEC1

GTZC_PERIPH_TZSC1

GTZC_PERIPH_TZIC1

GTZC_PERIPH_OCTOSPI1_MEM

GTZC_PERIPH_FSMC_MEM

GTZC_PERIPH_BKPSRAM

GTZC_PERIPH_OCTOSPI2_MEM

GTZC_PERIPH_HSPI1_MEM

GTZC_PERIPH_SRAM6

GTZC_PERIPH_MPCBB6_REG

GTZC_PERIPH_SRAM1

GTZC_PERIPH_MPCBB1_REG

GTZC_PERIPH_SRAM2

GTZC_PERIPH_MPCBB2_REG

GTZC_PERIPH_SRAM3

GTZC_PERIPH_MPCBB3_REG

GTZC_PERIPH_SRAM5

GTZC_PERIPH_MPCBB5_REG

GTZC_PERIPH_SPI3

GTZC_PERIPH_LPUART1

GTZC_PERIPH_I2C3

GTZC_PERIPH_LPTIM1

GTZC_PERIPH_LPTIM3

GTZC_PERIPH_LPTIM4

GTZC_PERIPH_OPAMP

GTZC_PERIPH_COMP

GTZC_PERIPH_ADC4

GTZC_PERIPH_VREFBUF

GTZC_PERIPH_DAC1

GTZC_PERIPH_ADF1

GTZC_PERIPH_SYSCFG

GTZC_PERIPH_RTC

GTZC_PERIPH_TAMP

GTZC_PERIPH_PWR

GTZC_PERIPH_RCC

GTZC_PERIPH_LPDMA1

GTZC_PERIPH_EXTI

GTZC_PERIPH_TZSC2

GTZC_PERIPH_TZIC2

GTZC_PERIPH_SRAM4

GTZC_PERIPH_MPCBB4_REG

GTZC_PERIPH_ALL

GTZC_TZSC_PERIPH_NUMBER

GTZC_TZIC_PERIPH_NUMBER

35 HAL HASH Generic Driver

35.1 HASH Firmware driver registers structures

35.1.1 HASH_InitTypeDef

HASH_InitTypeDef is defined in the stm32u5xx_hal_hash.h

Data Fields

- ***uint32_t DataType***
- ***uint32_t KeySize***
- ***uint8_t * pKey***

Field Documentation

- ***uint32_t HASH_InitTypeDef::DataType***
  32-bit data, 16-bit data, 8-bit data or 1-bit data. This parameter can be a value of ***HASH_Data_Type***.
- ***uint32_t HASH_InitTypeDef::KeySize***
  The key size is used only in HMAC operation.
- ***uint8_t* HASH_InitTypeDef::pKey***
  The key is used only in HMAC operation.

35.1.2 __HASH_HandleTypeDef

__HASH_HandleTypeDef is defined in the stm32u5xx_hal_hash.h

Data Fields

- ***HASH_InitTypeDef Init***
- ***uint8_t * pHashInBuffPtr***
- ***uint8_t * pHashOutBuffPtr***
- ***uint8_t * pHashKeyBuffPtr***
- ***uint8_t * pHashMsgBuffPtr***
- ***uint32_t HashBuffSize***
- ***__IO uint32_t HashInCount***
- ***__IO uint32_t HashITCounter***
- ***__IO uint32_t HashKeyCount***
- ***HAL_StatusTypeDef Status***
- ***HAL_HASH_PhaseTypeDef Phase***
- ***DMA_HandleTypeDef * hdmain***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_HASH_StateTypeDef State***
- ***HAL_HASH_SuspendTypeDef SuspendRequest***
- ***FlagStatus DigestCalculationDisable***
- ***__IO uint32_t NbWordsAlreadyPushed***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Accumulation***
- ***void(* InCpltCallback***
- ***void(* DgstCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HASH_InitTypeDef __HASH_HandleTypeDef::Init***
  HASH required parameters

- ***uint8_t* __HASH_HandleTypeDef::pHashInBuffPtr***
  Pointer to input buffer
- ***uint8_t* __HASH_HandleTypeDef::pHashOutBuffPtr***
  Pointer to output buffer (digest)
- ***uint8_t* __HASH_HandleTypeDef::pHashKeyBuffPtr***
  Pointer to key buffer (HMAC only)
- ***uint8_t* __HASH_HandleTypeDef::pHashMsgBuffPtr***
  Pointer to message buffer (HMAC only)
- ***uint32_t __HASH_HandleTypeDef::HashBuffSize***
  Size of buffer to be processed
- ***__IO uint32_t __HASH_HandleTypeDef::HashInCount***
  Counter of inputted data
- ***__IO uint32_t __HASH_HandleTypeDef::HashITCounter***
  Counter of issued interrupts
- ***__IO uint32_t __HASH_HandleTypeDef::HashKeyCount***
  Counter for Key inputted data (HMAC only)
- ***HAL_StatusTypeDef __HASH_HandleTypeDef::Status***
  HASH peripheral status
- ***HAL_HASH_PhaseTypeDef __HASH_HandleTypeDef::Phase***
  HASH peripheral phase
- ***DMA_HandleTypeDef* __HASH_HandleTypeDef::hdmain***
  HASH In DMA Handle parameters
- ***HAL_LockTypeDef __HASH_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_HASH_StateTypeDef __HASH_HandleTypeDef::State***
  HASH peripheral state
- ***HAL_HASH_SuspendTypeDef __HASH_HandleTypeDef::SuspendRequest***
  HASH peripheral suspension request flag
- ***FlagStatus __HASH_HandleTypeDef::DigestCalculationDisable***
  Digest calculation phase skip (MDMAT bit control) for multi-buffers DMA-based HMAC computation
- ***__IO uint32_t __HASH_HandleTypeDef::NbWordsAlreadyPushed***
  Numbers of words already pushed in FIFO before inputting new block
- ***__IO uint32_t __HASH_HandleTypeDef::ErrorCode***
  HASH Error code
- ***__IO uint32_t __HASH_HandleTypeDef::Accumulation***
  HASH multi buffers accumulation flag
- ***void(* __HASH_HandleTypeDef::InCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH input completion callback
- ***void(* __HASH_HandleTypeDef::DgstCpltCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH digest computation completion callback
- ***void(* __HASH_HandleTypeDef::ErrorCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH error callback
- ***void(* __HASH_HandleTypeDef::MspInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp Init callback
- ***void(* __HASH_HandleTypeDef::MspDeInitCallback)(struct __HASH_HandleTypeDef *hhash)***
  HASH Msp DeInit callback

35.2 HASH Firmware driver API description

The following section lists the various functions of the HASH library.

35.2.1 How to use this driver

The HASH HAL driver can be used as follows:

1. Initialize the HASH low level resources by implementing the HAL_HASH_MspInit():
   a. Enable the HASH interface clock using __HASH_CLK_ENABLE()
   b. When resorting to interrupt-based APIs (e.g. HAL_HASH_xxx_Start_IT())
      - Configure the HASH interrupt priority using HAL_NVIC_SetPriority()
      - Enable the HASH IRQ handler using HAL_NVIC_EnableIRQ()
      - In HASH IRQ handler, call HAL_HASH_IRQHandler() API
   c. When resorting to DMA-based APIs (e.g. HAL_HASH_xxx_Start_DMA())
      - Enable the DMAx interface clock using __DMAx_CLK_ENABLE()
      - Configure and enable one DMA channel to manage data transfer from memory to peripheral (input channel). Managing data transfer from peripheral to memory can be performed only using CPU.
      - Associate the initialized DMA handle to the HASH DMA handle using __HAL_LINKDMA()
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA channel: use HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ()
2. Initialize the HASH HAL using HAL_HASH_Init(). This function:
   a. resorts to HAL_HASH_MspInit() for low-level initialization,
   b. configures the data type: 1-bit, 8-bit, 16-bit or 32-bit.
3. Three processing schemes are available:
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASH_xxx_Start() for HASH or HAL_HMAC_xxx_Start() for HMAC
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASH_xxx_Start_IT() for HASH or HAL_HMAC_xxx_Start_IT() for HMAC
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASH_xxx_Start_DMA() for HASH or HAL_HMAC_xxx_Start_DMA() for HMAC. Note that in DMA mode, a call to HAL_HASH_xxx_Finish() is then required to retrieve the digest.
4. When the processing function is called after HAL_HASH_Init(), the HASH peripheral is initialized and processes the buffer fed in input. When the input data have all been fed to the Peripheral, the digest computation can start.
5. Multi-buffer processing is possible in polling, interrupt and DMA modes.
   a. In polling mode, only multi-buffer HASH processing is possible. API HAL_HASH_xxx_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_xxx_Accumulate_End() to enter the last one and retrieve as well the computed digest.
   b. In interrupt mode, API HAL_HASH_xxx_Accumulate_IT() must be called for each input buffer, except for the last one. User must resort to HAL_HASH_xxx_Accumulate_End_IT() to enter the last one and retrieve as well the computed digest.
   c. In DMA mode, multi-buffer HASH and HMAC processing are possible.
      - HASH processing: once initialization is done, MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the Peripheral through HAL_HASH_xxx_Start_DMA() API. Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASH_xxx_Start_DMA(). The digest can then be retrieved with a call to API HAL_HASH_xxx_Finish().
      - HMAC processing (requires to resort to extended functions): after initialization, the key and the first input buffer are entered in the Peripheral with the API HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL_HMACEx_xxx_Step2_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL_HMACEx_xxx_Step2_3_DMA(). The digest can finally be retrieved with a call to API HAL_HASH_xxx_Finish().

6. Context swapping.
   a. Two APIs are available to suspend HASH or HMAC processing:
      - HAL_HASH_SwFeed_ProcessSuspend() when data are entered by software (polling or IT mode),
      - HAL_HASH_DMAFeed_ProcessSuspend() when data are entered by DMA.
   b. When HASH or HMAC processing is suspended, HAL_HASH_ContextSaving() allows to save in memory the Peripheral context. This context can be restored afterwards to resume the HASH processing thanks to HAL_HASH_ContextRestoring().
   c. Once the HASH Peripheral has been restored to the same configuration as that at suspension time, processing can be restarted with the same API call (same API, same handle, same parameters) as done before the suspension. Relevant parameters to restart at the proper location are internally saved in the HASH handle.
7. Call HAL_HASH_DeInit() to deinitialize the HASH peripheral.

Remarks on message length

1. HAL in interruption mode (interruptions driven)
   a. Due to HASH peripheral hardware design, the peripheral interruption is triggered every 64 bytes. This is why, for driver implementation simplicity's sake, user is requested to enter a message the length of which is a multiple of 4 bytes.
   b. When the message length (in bytes) is not a multiple of words, a specific field exists in HASH_STR to specify which bits to discard at the end of the complete message to process only the message bits and not extra bits.
   c. If user needs to perform a hash computation of a large input buffer that is spread around various places in memory and where each piece of this input buffer is not necessarily a multiple of 4 bytes in size, it becomes necessary to use a temporary buffer to format the data accordingly before feeding them to the Peripheral. It is advised to the user to
      - achieve the first formatting operation by software then enter the data
      - while the Peripheral is processing the first input set, carry out the second formatting operation by software, to be ready when DINIS occurs.
      - repeat step 2 until the whole message is processed.
1. HAL in DMA mode
   a. Again, due to hardware design, the DMA transfer to feed the data can only be done on a word-basis. The same field described above in HASH_STR is used to specify which bits to discard at the end of the DMA transfer to process only the message bits and not extra bits. Due to hardware implementation, this is possible only at the end of the complete message. When several DMA transfers are needed to enter the message, this is not applicable at the end of the intermediary transfers.
   b. Similarly to the interruption-driven mode, it is suggested to the user to format the consecutive chunks of data by software while the DMA transfer and processing is on-going for the first parts of the message. Due to the 32-bit alignment required for the DMA transfer, it is underlined that the software formatting operation is more complex than in the IT mode.

Callback registration

1. The compilation define USE_HAL_HASH_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_HASH_RegisterCallback() to register a user callback.
2. Function HAL_HASH_RegisterCallback() allows to register following callbacks: (+) InCpltCallback : callback for input completion. (+) DgstCpltCallback : callback for digest computation completion. (+) ErrorCallback : callback for error. (+) MspInitCallback : HASH MspInit. (+) MspDeInitCallback : HASH MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
3. Use function HAL_HASH_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_HASH_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) InCpltCallback : callback for input completion. (+) DgstCpltCallback : callback for digest computation completion. (+) ErrorCallback : callback for error. (+) MspInitCallback : HASH MspInit. (+) MspDeInitCallback : HASH MspDeInit.

4. By default, after the HAL_HASH_Init and if the state is HAL_HASH_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_HASH_InCpltCallback(), HAL_HASH_DgstCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_HASH_Init and HAL_HASH_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the HAL_HASH_Init and HAL_HASH_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_HASH_RegisterCallback before calling HAL_HASH_DeInit or HAL_HASH_Init function. When The compilation define USE_HAL_HASH_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

(#) The compilation define USE_HAL_HASH_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_HASH_RegisterCallback() to register a user callback. (#) Function HAL_HASH_RegisterCallback() allows to register following callbacks:

- InCpltCallback : callback for input completion.
- DgstCpltCallback : callback for digest computation completion.
- ErrorCallback : callback for error.
- MspInitCallback : HASH MspInit.
- MspDeInitCallback : HASH MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_HASH_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_HASH_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- InCpltCallback : callback for input completion.
- DgstCpltCallback : callback for digest computation completion.
- ErrorCallback : callback for error.
- MspInitCallback : HASH MspInit.
- MspDeInitCallback : HASH MspDeInit. (#) By default, after the HAL_HASH_Init and if the state is HAL_HASH_STATE_RESET all callbacks are reset to the corresponding legacy weak (surcharged) functions: examples HAL_HASH_InCpltCallback(), HAL_HASH_DgstCpltCallback() Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (surcharged) functions in the HAL_HASH_Init and HAL_HASH_DeInit only when these callbacks are null (not registered beforehand) If not, MspInit or MspDeInit are not null, the HAL_HASH_Init and HAL_HASH_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand). Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_HASH_RegisterCallback before calling HAL_HASH_DeInit or HAL_HASH_Init function. When The compilation define USE_HAL_HASH_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

35.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the HASH according to the specified parameters in the HASH_InitTypeDef and create the associated handle
- DeInitialize the HASH peripheral
- Initialize the HASH MCU Specific Package (MSP)
- DeInitialize the HASH MSP

This section provides as well call back functions definitions for user code to manage:

- Input data transfer to Peripheral completion
- Calculated digest retrieval completion
- Error management

This section contains the following APIs:

- *HAL_HASH_Init()*
- *HAL_HASH_DeInit()*

- *HAL_HASH_MspInit()*
- *HAL_HASH_MspDeInit()*
- *HAL_HASH_InCpltCallback()*
- *HAL_HASH_DgstCpltCallback()*
- *HAL_HASH_ErrorCallback()*
- *HAL_HASH_RegisterCallback()*
- *HAL_HASH_UnRegisterCallback()*

35.2.3 Polling mode HASH processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start()
  - HAL_HASH_MD5_Accmlt()
  - HAL_HASH_MD5_Accmlt_End()
- SHA1
  - HAL_HASH_SHA1_Start()
  - HAL_HASH_SHA1_Accmlt()
  - HAL_HASH_SHA1_Accmlt_End()

For a single buffer to be hashed, user can resort to HAL_HASH_xxx_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to HAL_HASH_xxx_Accumulate() and wrap-up the digest computation by a call to HAL_HASH_xxx_Accumulate_End().

This section contains the following APIs:

- *HAL_HASH_MD5_Start()*
- *HAL_HASH_MD5_Accmlt()*
- *HAL_HASH_MD5_Accmlt_End()*
- *HAL_HASH_SHA1_Start()*
- *HAL_HASH_SHA1_Accmlt()*
- *HAL_HASH_SHA1_Accmlt_End()*

35.2.4 Interruption mode HASH processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start_IT()
  - HAL_HASH_MD5_Accmlt_IT()
  - HAL_HASH_MD5_Accmlt_End_IT()
- SHA1
  - HAL_HASH_SHA1_Start_IT()
  - HAL_HASH_SHA1_Accmlt_IT()
  - HAL_HASH_SHA1_Accmlt_End_IT()

API HAL_HASH_IRQHandler() manages each HASH interruption.

Note that HAL_HASH_IRQHandler() manages as well HASH Peripheral interruptions when in HMAC processing mode.

This section contains the following APIs:

- *HAL_HASH_MD5_Start_IT()*
- *HAL_HASH_MD5_Accmlt_IT()*
- *HAL_HASH_MD5_Accmlt_End_IT()*
- *HAL_HASH_SHA1_Start_IT()*

- *HAL_HASH_SHA1_Accmlt_IT()*
- *HAL_HASH_SHA1_Accmlt_End_IT()*
- *HAL_HASH_IRQHandler()*

35.2.5 DMA mode HASH processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- MD5
  - HAL_HASH_MD5_Start_DMA()
  - HAL_HASH_MD5_Finish()
- SHA1
  - HAL_HASH_SHA1_Start_DMA()
  - HAL_HASH_SHA1_Finish()

When resorting to DMA mode to enter the data in the Peripheral, user must resort to HAL_HASH_xxx_Start_DMA() then read the resulting digest with HAL_HASH_xxx_Finish().

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to HAL_HASH_xxx_Start_DMA(). Then, MDMAT bit needs to be reset before the last call to HAL_HASH_xxx_Start_DMA(). Digest is finally retrieved thanks to HAL_HASH_xxx_Finish().

This section contains the following APIs:

- *HAL_HASH_MD5_Start_DMA()*
- *HAL_HASH_MD5_Finish()*
- *HAL_HASH_SHA1_Start_DMA()*
- *HAL_HASH_SHA1_Finish()*

35.2.6 Polling mode HMAC processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start()
- SHA1
  - HAL_HMAC_SHA1_Start()

This section contains the following APIs:

- *HAL_HMAC_MD5_Start()*
- *HAL_HMAC_SHA1_Start()*

35.2.7 Interrupt mode HMAC processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start_IT()
- SHA1
  - HAL_HMAC_SHA1_Start_IT()

This section contains the following APIs:

- *HAL_HMAC_MD5_Start_IT()*
- *HAL_HMAC_SHA1_Start_IT()*

35.2.8 DMA mode HMAC processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- MD5
  - HAL_HMAC_MD5_Start_DMA()
- SHA1
  - HAL_HMAC_SHA1_Start_DMA()

When resorting to DMA mode to enter the data in the Peripheral for HMAC processing, user must resort to HAL_HMAC_xxx_Start_DMA() then read the resulting digest with HAL_HASH_xxx_Finish().

This section contains the following APIs:

- *HAL_HMAC_MD5_Start_DMA()*
- *HAL_HMAC_SHA1_Start_DMA()*

35.2.9 Peripheral State methods

This section permits to get in run-time the state and the peripheral handle status of the peripheral:

- HAL_HASH_GetState()
- HAL_HASH_GetStatus()

Additionally, this subsection provides functions allowing to save and restore the HASH or HMAC processing context in case of calculation suspension:

- HAL_HASH_ContextSaving()
- HAL_HASH_ContextRestoring()

This subsection provides functions allowing to suspend the HASH processing

- when input are fed to the Peripheral by software
  - HAL_HASH_SwFeed_ProcessSuspend()
- when input are fed to the Peripheral by DMA
  - HAL_HASH_DMAFeed_ProcessSuspend()

This section contains the following APIs:

- *HAL_HASH_GetState()*
- *HAL_HASH_GetStatus()*
- *HAL_HASH_ContextSaving()*
- *HAL_HASH_ContextRestoring()*
- *HAL_HASH_SwFeed_ProcessSuspend()*
- *HAL_HASH_DMAFeed_ProcessSuspend()*
- *HAL_HASH_GetError()*

35.2.10 Detailed description of functions

HAL_HASH_Init

Function name

HAL_StatusTypeDef HAL_HASH_Init (HASH_HandleTypeDef * hhash)

Function description

Initialize the HASH according to the specified parameters in the HASH_HandleTypeDef and create the associated handle.

Parameters

- **hhash:** HASH handle

Return values

- **HAL:** status

Notes

- Only MDMAT and DATATYPE bits of HASH Peripheral are set by HAL_HASH_Init(), other configuration bits are set by HASH or HMAC processing APIs.
- MDMAT bit is systematically reset by HAL_HASH_Init(). To set it for multi-buffer HASH processing, user needs to resort to __HAL_HASH_SET_MDMAT() macro. For HMAC multi-buffer processing, the relevant APIs manage themselves the MDMAT bit.

HAL_HASH_DeInit

Function name

HAL_StatusTypeDef HAL_HASH_DeInit (HASH_HandleTypeDef * hhash)

Function description

DeInitialize the HASH peripheral.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** status

HAL_HASH_MspInit

Function name

void HAL_HASH_MspInit (HASH_HandleTypeDef * hhash)

Function description

Initialize the HASH MSP.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

HAL_HASH_MspDeInit

Function name

void HAL_HASH_MspDeInit (HASH_HandleTypeDef * hhash)

Function description

DeInitialize the HASH MSP.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

HAL_HASH_InCpltCallback

Function name

void HAL_HASH_InCpltCallback (HASH_HandleTypeDef * hhash)

Function description

Input data transfer complete call back.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_InCpltCallback() is called when the complete input message has been fed to the Peripheral. This API is invoked only when input data are entered under interruption or through DMA.
- In case of HASH or HMAC multi-buffer DMA feeding case (MDMAT bit set), HAL_HASH_InCpltCallback() is called at the end of each buffer feeding to the Peripheral.

HAL_HASH_DgstCpltCallback

Function name

void HAL_HASH_DgstCpltCallback (HASH_HandleTypeDef * hhash)

Function description

Digest computation complete call back.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_DgstCpltCallback() is used under interruption, is not relevant with DMA.

HAL_HASH_ErrorCallback

Function name

void HAL_HASH_ErrorCallback (HASH_HandleTypeDef * hhash)

Function description

Error callback.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- Code user can resort to hhash->Status (HAL_ERROR, HAL_TIMEOUT,...) to retrieve the error type.

HAL_HASH_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_RegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID, pHASH_CallbackTypeDef pCallback)

Function description

Register a User HASH Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hhash:** HASH handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_HASH_INPUTCPLT_CB_ID HASH input completion Callback ID
  – HAL_HASH_DGSTCPLT_CB_ID HASH digest computation completion Callback ID
  – HAL_HASH_ERROR_CB_ID HASH error Callback ID
  – HAL_HASH_MSPINIT_CB_ID HASH MspInit callback ID
  – HAL_HASH_MSPDEINIT_CB_ID HASH MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_HASH_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HASH_UnRegisterCallback (HASH_HandleTypeDef * hhash, HAL_HASH_CallbackIDTypeDef CallbackID)

Function description

Unregister a HASH Callback HASH Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hhash:** HASH handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_HASH_INPUTCPLT_CB_ID HASH input completion Callback ID
  – HAL_HASH_DGSTCPLT_CB_ID HASH digest computation completion Callback ID
  – HAL_HASH_ERROR_CB_ID HASH error Callback ID
  – HAL_HASH_MSPINIT_CB_ID HASH MspInit callback ID
  – HAL_HASH_MSPDEINIT_CB_ID HASH MspDeInit callback ID

Return values

- **status:**

HAL_HASH_SHA1_Start

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Start

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Accmlt

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_MD5_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_MD5_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASH_MD5_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_MD5_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_SHA1_Accmlt

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_SHA1_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_SHA1_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASH_SHA1_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_SHA1_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_MD5_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_MD5_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_SHA1_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_SHA1_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_SHA1_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in SHA1 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_SHA1_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA1 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_SHA1_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_SHA1_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_SHA1_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_SHA1_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_SHA1_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Start_IT

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in MD5 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_MD5_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in MD5 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASH_MD5_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASH_MD5_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASH_MD5_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASH_MD5_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASH_MD5_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASH_IRQHandler

Function name

void HAL_HASH_IRQHandler (HASH_HandleTypeDef * hhash)

Function description

Handle HASH interrupt request.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- HAL_HASH_IRQHandler() handles interrupts in HMAC processing as well.
- In case of error reported during the HASH interruption processing, HAL_HASH_ErrorCallback() API is called so that user code can manage the error. The error type is available in hhash->Status field.

HAL_HASH_SHA1_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA1 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASH_SHA1_Finish() API must be called to retrieve the computed digest.

HAL_HASH_SHA1_Finish

Function name

HAL_StatusTypeDef HAL_HASH_SHA1_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA1 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASH_SHA1_Finish() can be used as well to retrieve the digest in HMAC SHA1 mode.

HAL_HASH_MD5_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in MD5 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASH_MD5_Finish() API must be called to retrieve the computed digest.

HAL_HASH_MD5_Finish

Function name

HAL_StatusTypeDef HAL_HASH_MD5_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in MD5 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASH_MD5_Finish() can be used as well to retrieve the digest in HMAC MD5 mode.

HAL_HMAC_SHA1_Start

Function name

HAL_StatusTypeDef HAL_HMAC_SHA1_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_MD5_Start

Function name

HAL_StatusTypeDef HAL_HMAC_MD5_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_MD5_Start_IT

Function name

HAL_StatusTypeDef HAL_HMAC_MD5_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC MD5 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 16 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_SHA1_Start_IT

Function name

HAL_StatusTypeDef HAL_HMAC_SHA1_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC SHA1 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 20 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMAC_SHA1_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMAC_SHA1_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA1 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASH_SHA1_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMAC_MD5_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMAC_MD5_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC MD5 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASH_MD5_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HASH_GetState

Function name

HAL_HASH_StateTypeDef HAL_HASH_GetState (HASH_HandleTypeDef * hhash)

Function description

Return the HASH handle state.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** HASH state

Notes

- The API yields the current state of the handle (BUSY, READY,...).

HAL_HASH_GetStatus

Function name

HAL_StatusTypeDef HAL_HASH_GetStatus (HASH_HandleTypeDef * hhash)

Function description

Return the HASH HAL status.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** status

Notes

- The API yields the HAL status of the handle: it is the result of the latest HASH processing and allows to report any issue (e.g. HAL_TIMEOUT).

HAL_HASH_ContextSaving

Function name

void HAL_HASH_ContextSaving (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

Save the HASH context in case of processing suspension.

Parameters

- **hhash:** HASH handle.
- **pMemBuffer:** pointer to the memory buffer where the HASH context is saved.

Return values

- **None:**

Notes

- The IMR, STR, CR then all the CSR registers are saved in that order. Only the r/w bits are read to be restored later on.
- By default, all the context swap registers (there are HASH_NUMBER_OF_CSR_REGISTERS of those) are saved.
- pMemBuffer points to a buffer allocated by the user. The buffer size must be at least (HASH_NUMBER_OF_CSR_REGISTERS + 3) * 4 uint8 long.

HAL_HASH_ContextRestoring

Function name

void HAL_HASH_ContextRestoring (HASH_HandleTypeDef * hhash, uint8_t * pMemBuffer)

Function description

Restore the HASH context in case of processing resumption.

Parameters

- **hhash:** HASH handle.
- **pMemBuffer:** pointer to the memory buffer where the HASH context is stored.

Return values

- **None:**

Notes

- The IMR, STR, CR then all the CSR registers are restored in that order. Only the r/w bits are restored.
- By default, all the context swap registers (HASH_NUMBER_OF_CSR_REGISTERS of those) are restored (all of them have been saved by default beforehand).

HAL_HASH_SwFeed_ProcessSuspend

Function name

void HAL_HASH_SwFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)

Function description

Initiate HASH processing suspension when in polling or interruption mode.

Parameters

- **hhash:** HASH handle.

Return values

- **None:**

Notes

- Set the handle field SuspendRequest to the appropriate value so that the on-going HASH processing is suspended as soon as the required conditions are met. Note that the actual suspension is carried out by the functions HASH_WriteData() in polling mode and HASH_IT() in interruption mode.

HAL_HASH_DMAFeed_ProcessSuspend

Function name

HAL_StatusTypeDef HAL_HASH_DMAFeed_ProcessSuspend (HASH_HandleTypeDef * hhash)

Function description

Suspend the HASH processing when in DMA mode.

Parameters

- **hhash:** HASH handle.

Return values

- **HAL:** status

Notes

- When suspension attempt occurs at the very end of a DMA transfer and all the data have already been entered in the Peripheral, hhash->State is set to HAL_HASH_STATE_READY and the API returns HAL_ERROR. It is recommended to wrap-up the processing in reading the digest as usual.

HAL_HASH_GetError

Function name

uint32_t HAL_HASH_GetError (HASH_HandleTypeDef * hhash)

Function description

Return the HASH handle error code.

Parameters

- **hhash:** pointer to a HASH_HandleTypeDef structure.

Return values

- **HASH:** Error Code

HASH_Start

Function name

HAL_StatusTypeDef HASH_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout, uint32_t Algorithm)

Function description

Initialize the HASH peripheral, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** Timeout value.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HASH_Accumulate

Function name

HAL_StatusTypeDef HASH_Accumulate (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

If not already done, initialize the HASH peripheral then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HASH_Accumulate_IT

Function name

HAL_StatusTypeDef HASH_Accumulate_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

If not already done, initialize the HASH peripheral then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HASH_Start_IT

Function name

HAL_StatusTypeDef HASH_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Algorithm)

Function description

Initialize the HASH peripheral, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HASH_Start_DMA

Function name

HAL_StatusTypeDef HASH_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

Initialize the HASH peripheral then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HASH_Finish

Function name

HAL_StatusTypeDef HASH_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.

HMAC_Start

Function name

HAL_StatusTypeDef HMAC_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout, uint32_t Algorithm)

Function description

Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Timeout:** Timeout value.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HMAC_Start_IT

Function name

HAL_StatusTypeDef HMAC_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Algorithm)

Function description

Initialize the HASH peripheral in HMAC mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HMAC_Start_DMA

Function name

HAL_StatusTypeDef HMAC_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint32_t Algorithm)

Function description

Initialize the HASH peripheral in HMAC mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **Algorithm:** HASH algorithm.

Return values

- **HAL:** status

Notes

- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- In case of multi-buffer HMAC processing, the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only the length of the last buffer of the thread doesn't have to be a multiple of 4.

35.3 HASH Firmware driver defines

The following section lists the various define and macros of the module.

35.3.1 HASH

HASH

HASH algorithm mode

HASH_ALGOMODE_HASH

Algorithm is HASH

HASH_ALGOMODE_HMAC

Algorithm is HMAC

HASH algorithm selection

HASH_ALGOSELECTION_SHA1

HASH function is SHA1

HASH_ALGOSELECTION_MD5

HASH function is MD5

HASH_ALGOSELECTION_SHA224

HASH function is SHA224

HASH_ALGOSELECTION_SHA256

HASH function is SHA256

HASH input data type

HASH_DATATYPE_32B

32-bit data. No swapping

HASH_DATATYPE_16B

16-bit data. Each half word is swapped

HASH_DATATYPE_8B

8-bit data. All bytes are swapped

HASH_DATATYPE_1B

1-bit data. In the word all bits are swapped

HASH Digest Calculation Status

HASH_DIGEST_CALCULATION_NOT_STARTED

DCAL not set after input data written in DIN register

HASH_DIGEST_CALCULATION_STARTED

DCAL set after input data written in DIN register

HASH DMA suspension words limit

HASH_DMA_SUSPENSION_WORDS_LIMIT

Number of words below which DMA suspension is aborted

HASH Error Definition

HAL_HASH_ERROR_NONE

No error

HAL_HASH_ERROR_IT

IT-based process error

HAL_HASH_ERROR_DMA

DMA-based process error

HAL_HASH_ERROR_INVALID_CALLBACK

Invalid Callback error

HASH Exported Macros

__HAL_HASH_GET_FLAG

Description:
- Check whether or not the specified HASH flag is set.

Parameters:
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete.
  - HASH_FLAG_DMAS DMA interface is enabled (DMAE=1) or a transfer is ongoing.
  - HASH_FLAG_BUSY The hash core is Busy : processing a block of data.
  - HASH_FLAG_DINNE DIN not empty : the input buffer contains at least one word of data.

Return value:
- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_HASH_CLEAR_FLAG

Description:
- Clear the specified HASH flag.

Parameters:
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - HASH_FLAG_DINIS A new block can be entered into the input buffer.
  - HASH_FLAG_DCIS Digest calculation complete

Return value:
- None

__HAL_HASH_ENABLE_IT

Description:
- Enable the specified HASH interrupt.

Parameters:
- __INTERRUPT__: specifies the HASH interrupt source to enable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:
- None

__HAL_HASH_DISABLE_IT

Description:
- Disable the specified HASH interrupt.

Parameters:
- __INTERRUPT__: specifies the HASH interrupt source to disable. This parameter can be one of the following values:
  - HASH_IT_DINI A new block can be entered into the input buffer (DIN)
  - HASH_IT_DCI Digest calculation complete

Return value:
- None

__HAL_HASH_RESET_HANDLE_STATE

Description:
- Reset HASH handle state.

Parameters:
- __HANDLE__: HASH handle.

Return value:
- None

__HAL_HASH_RESET_HANDLE_STATUS

Description:
- Reset HASH handle status.

Parameters:
- __HANDLE__: HASH handle.

Return value:
- None

__HAL_HASH_SET_MDMAT

Description:
- Enable the multi-buffer DMA transfer mode.

Return value:
- None

Notes:
- This bit is set when hashing large files when multiple DMA transfers are needed.

__HAL_HASH_RESET_MDMAT

Description:
- Disable the multi-buffer DMA transfer mode.

Return value:
- None

__HAL_HASH_START_DIGEST

Description:
- Start the digest computation.

Return value:
- None

__HAL_HASH_SET_NBVALIDBITS

Description:
- Set the number of valid bits in the last word written in data register DIN.

Parameters:
- __SIZE__: size in bytes of last data written in Data register.

Return value:
- None

__HAL_HASH_INIT

Description:
- Reset the HASH core.

Return value:
- None

HASH flags definitions

HASH_FLAG_DINIS

16 locations are free in the DIN : a new block can be entered in the Peripheral

HASH_FLAG_DCIS

Digest calculation complete

HASH_FLAG_DMAS

DMA interface is enabled (DMAE=1) or a transfer is ongoing

HASH_FLAG_BUSY

The hash core is Busy, processing a block of data

HASH_FLAG_DINNE

DIN not empty : the input buffer contains at least one word of data

HMAC key length type

HASH_HMAC_KEYTYPE_SHORTKEY

HMAC Key size is <= 64 bytes

HASH_HMAC_KEYTYPE_LONGKEY

HMAC Key size is > 64 bytes

HASH interrupts definitions

HASH_IT_DINI

A new block can be entered into the input buffer (DIN)

HASH_IT_DCI

Digest calculation complete

HASH Number of Context Swap Registers

HASH_NUMBER_OF_CSR_REGISTERS

Number of Context Swap Registers

HASH TimeOut Value

HASH_TIMEOUTVALUE

Time-out value

36 HAL HASH Extension Driver

36.1 HASHEx Firmware driver API description

The following section lists the various functions of the HASHEx library.

36.1.1 HASH peripheral extended features

The SHA-224 and SHA-256 HASH and HMAC processing can be carried out exactly the same way as for SHA-1 or MD-5 algorithms.

1. Three modes are available.
   a. Polling mode: processing APIs are blocking functions i.e. they process the data and wait till the digest computation is finished, e.g. HAL_HASHEx_xxx_Start()
   b. Interrupt mode: processing APIs are not blocking functions i.e. they process the data under interrupt, e.g. HAL_HASHEx_xxx_Start_IT()
   c. DMA mode: processing APIs are not blocking functions and the CPU is not used for data transfer i.e. the data transfer is ensured by DMA, e.g. HAL_HASHEx_xxx_Start_DMA(). Note that in DMA mode, a call to HAL_HASHEx_xxx_Finish() is then required to retrieve the digest.
2. Multi-buffer processing is possible in polling, interrupt and DMA modes.
   a. In polling mode, only multi-buffer HASH processing is possible. API HAL_HASHEx_xxx_Accumulate() must be called for each input buffer, except for the last one. User must resort to HAL_HASHEx_xxx_Accumulate_End() to enter the last one and retrieve as well the computed digest.
   b. In interrupt mode, API HAL_HASHEx_xxx_Accumulate_IT() must be called for each input buffer, except for the last one. User must resort to HAL_HASHEx_xxx_Accumulate_End_IT() to enter the last one and retrieve as well the computed digest.
   c. In DMA mode, multi-buffer HASH and HMAC processing are possible.
      - HASH processing: once initialization is done, MDMAT bit must be set through __HAL_HASH_SET_MDMAT() macro. From that point, each buffer can be fed to the Peripheral through HAL_HASHEx_xxx_Start_DMA() API. Before entering the last buffer, reset the MDMAT bit with __HAL_HASH_RESET_MDMAT() macro then wrap-up the HASH processing in feeding the last input buffer through the same API HAL_HASHEx_xxx_Start_DMA(). The digest can then be retrieved with a call to API HAL_HASHEx_xxx_Finish().
      - HMAC processing (MD-5, SHA-1, SHA-224 and SHA-256 must all resort to extended functions): after initialization, the key and the first input buffer are entered in the Peripheral with the API HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and starts step 2. The following buffers are next entered with the API HAL_HMACEx_xxx_Step2_DMA(). At this point, the HMAC processing is still carrying out step 2. Then, step 2 for the last input buffer and step 3 are carried out by a single call to HAL_HMACEx_xxx_Step2_3_DMA(). The digest can finally be retrieved with a call to API HAL_HASH_xxx_Finish() for MD-5 and SHA-1, to HAL_HASHEx_xxx_Finish() for SHA-224 and SHA-256.

36.1.2 Polling mode HASH extended processing functions

This section provides functions allowing to calculate in polling mode the hash value using one of the following algorithms:

- SHA224
  – HAL_HASHEx_SHA224_Start()
  – HAL_HASHEx_SHA224_Accmlt()
  – HAL_HASHEx_SHA224_Accmlt_End()
- SHA256
  – HAL_HASHEx_SHA256_Start()
  – HAL_HASHEx_SHA256_Accmlt()
  – HAL_HASHEx_SHA256_Accmlt_End()

For a single buffer to be hashed, user can resort to HAL_HASH_xxx_Start().

In case of multi-buffer HASH processing (a single digest is computed while several buffers are fed to the Peripheral), the user can resort to successive calls to HAL_HASHEx_xxx_Accumulate() and wrap-up the digest computation by a call to HAL_HASHEx_xxx_Accumulate_End().

This section contains the following APIs:

- *HAL_HASHEx_SHA224_Start()*
- *HAL_HASHEx_SHA224_Accmlt()*
- *HAL_HASHEx_SHA224_Accmlt_End()*
- *HAL_HASHEx_SHA256_Start()*
- *HAL_HASHEx_SHA256_Accmlt()*
- *HAL_HASHEx_SHA256_Accmlt_End()*

36.1.3 Interruption mode HASH extended processing functions

This section provides functions allowing to calculate in interrupt mode the hash value using one of the following algorithms:

- SHA224
  – HAL_HASHEx_SHA224_Start_IT()
  – HAL_HASHEx_SHA224_Accmlt_IT()
  – HAL_HASHEx_SHA224_Accmlt_End_IT()
- SHA256
  – HAL_HASHEx_SHA256_Start_IT()
  – HAL_HASHEx_SHA256_Accmlt_IT()
  – HAL_HASHEx_SHA256_Accmlt_End_IT()

This section contains the following APIs:

- *HAL_HASHEx_SHA224_Start_IT()*
- *HAL_HASHEx_SHA224_Accmlt_IT()*
- *HAL_HASHEx_SHA224_Accmlt_End_IT()*
- *HAL_HASHEx_SHA256_Start_IT()*
- *HAL_HASHEx_SHA256_Accmlt_IT()*
- *HAL_HASHEx_SHA256_Accmlt_End_IT()*

36.1.4 DMA mode HASH extended processing functions

This section provides functions allowing to calculate in DMA mode the hash value using one of the following algorithms:

- SHA224
  – HAL_HASHEx_SHA224_Start_DMA()
  – HAL_HASHEx_SHA224_Finish()
- SHA256
  – HAL_HASHEx_SHA256_Start_DMA()
  – HAL_HASHEx_SHA256_Finish()

When resorting to DMA mode to enter the data in the Peripheral, user must resort to HAL_HASHEx_xxx_Start_DMA() then read the resulting digest with HAL_HASHEx_xxx_Finish().

In case of multi-buffer HASH processing, MDMAT bit must first be set before the successive calls to HAL_HASHEx_xxx_Start_DMA(). Then, MDMAT bit needs to be reset before the last call to HAL_HASHEx_xxx_Start_DMA(). Digest is finally retrieved thanks to HAL_HASHEx_xxx_Finish().

This section contains the following APIs:

- *HAL_HASHEx_SHA224_Start_DMA()*
- *HAL_HASHEx_SHA224_Finish()*
- *HAL_HASHEx_SHA256_Start_DMA()*
- *HAL_HASHEx_SHA256_Finish()*

36.1.5 Polling mode HMAC extended processing functions

This section provides functions allowing to calculate in polling mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start()
- SHA256
  - HAL_HMACEx_SHA256_Start()

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start()***
- ***HAL_HMACEx_SHA256_Start()***

36.1.6 Interrupt mode HMAC extended processing functions

This section provides functions allowing to calculate in interrupt mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start_IT()
- SHA256
  - HAL_HMACEx_SHA256_Start_IT()

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start_IT()***
- ***HAL_HMACEx_SHA256_Start_IT()***

36.1.7 DMA mode HMAC extended processing functions

This section provides functions allowing to calculate in DMA mode the HMAC value using one of the following algorithms:

- SHA224
  - HAL_HMACEx_SHA224_Start_DMA()
- SHA256
  - HAL_HMACEx_SHA256_Start_DMA()

When resorting to DMA mode to enter the data in the Peripheral for HMAC processing, user must resort to HAL_HMACEx_xxx_Start_DMA() then read the resulting digest with HAL_HASHEx_xxx_Finish().

This section contains the following APIs:

- ***HAL_HMACEx_SHA224_Start_DMA()***
- ***HAL_HMACEx_SHA256_Start_DMA()***

36.1.8 Multi-buffer DMA mode HMAC extended processing functions

This section provides functions to manage HMAC multi-buffer DMA-based processing for MD5, SHA1, SHA224 and SHA256 algorithms.

- MD5
  - HAL_HMACEx_MD5_Step1_2_DMA()
  - HAL_HMACEx_MD5_Step2_DMA()
  - HAL_HMACEx_MD5_Step2_3_DMA()
- SHA1
  - HAL_HMACEx_SHA1_Step1_2_DMA()
  - HAL_HMACEx_SHA1_Step2_DMA()
  - HAL_HMACEx_SHA1_Step2_3_DMA()

- SHA256
  - HAL_HMACEx_SHA224_Step1_2_DMA()
  - HAL_HMACEx_SHA224_Step2_DMA()
  - HAL_HMACEx_SHA224_Step2_3_DMA()
- SHA256
  - HAL_HMACEx_SHA256_Step1_2_DMA()
  - HAL_HMACEx_SHA256_Step2_DMA()
  - HAL_HMACEx_SHA256_Step2_3_DMA()

User must first start-up the multi-buffer DMA-based HMAC computation in calling HAL_HMACEx_xxx_Step1_2_DMA(). This carries out HMAC step 1 and intiates step 2 with the first input buffer.

The following buffers are next fed to the Peripheral with a call to the API HAL_HMACEx_xxx_Step2_DMA(). There may be several consecutive calls to this API.

Multi-buffer DMA-based HMAC computation is wrapped up by a call to HAL_HMACEx_xxx_Step2_3_DMA(). This finishes step 2 in feeding the last input buffer to the Peripheral then carries out step 3.

Digest is retrieved by a call to HAL_HASH_xxx_Finish() for MD-5 or SHA-1, to HAL_HASHEx_xxx_Finish() for SHA-224 or SHA-256.

If only two buffers need to be consecutively processed, a call to HAL_HMACEx_xxx_Step1_2_DMA() followed by a call to HAL_HMACEx_xxx_Step2_3_DMA() is sufficient.

This section contains the following APIs:
- *HAL_HMACEx_MD5_Step1_2_DMA()*
- *HAL_HMACEx_MD5_Step2_DMA()*
- *HAL_HMACEx_MD5_Step2_3_DMA()*
- *HAL_HMACEx_SHA1_Step1_2_DMA()*
- *HAL_HMACEx_SHA1_Step2_DMA()*
- *HAL_HMACEx_SHA1_Step2_3_DMA()*
- *HAL_HMACEx_SHA224_Step1_2_DMA()*
- *HAL_HMACEx_SHA224_Step2_DMA()*
- *HAL_HMACEx_SHA224_Step2_3_DMA()*
- *HAL_HMACEx_SHA256_Step1_2_DMA()*
- *HAL_HMACEx_SHA256_Step2_DMA()*
- *HAL_HMACEx_SHA256_Step2_3_DMA()*

36.1.9 Detailed description of functions

HAL_HASHEx_SHA224_Start

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Accmlt

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA224_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA224_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA224_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA224_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA224_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA224_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Start

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Accmlt

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA256_Accmlt() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA256_Accmlt_End().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- Digest is not retrieved by this API, user must resort to HAL_HASHEx_SHA256_Accmlt_End() to read it, feeding at the same time the last input buffer to the Peripheral.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA256_Accmlt_End() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA256_Accmlt_End

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt_End (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA256_Accmlt() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Start_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in SHA224 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA224 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA224_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA224_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA224_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA224_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA224_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Start_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in SHA256 mode, next process pInBuffer then read the computed digest in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA256_Accmlt_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

If not already done, initialize the HASH peripheral in SHA256 mode then processes pInBuffer in interruption mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes, must be a multiple of 4.

Return values

- **HAL:** status

Notes

- Consecutive calls to HAL_HASHEx_SHA256_Accmlt_IT() can be used to feed several input buffers back-to-back to the Peripheral that will yield a single HASH signature once all buffers have been entered. Wrap-up of input buffers feeding and retrieval of digest is done by a call to HAL_HASHEx_SHA256_Accmlt_End_IT().
- Field hhash->Phase of HASH handle is tested to check whether or not the Peripheral has already been initialized.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. Only HAL_HASHEx_SHA256_Accmlt_End_IT() is able to manage the ending buffer with a length in bytes not a multiple of 4.

HAL_HASHEx_SHA256_Accmlt_End_IT

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Accmlt_End_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

End computation of a single HASH signature after several calls to HAL_HASHEx_SHA256_Accmlt_IT() API.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.

HAL_HASHEx_SHA224_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA224 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA224_Finish

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA224_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA224 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASHEx_SHA224_Finish() can be used as well to retrieve the digest in HMAC SHA224 mode.

HAL_HASHEx_SHA256_Start_DMA

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in SHA256 mode then initiate a DMA transfer to feed the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfer is finished, HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HASHEx_SHA256_Finish

Function name

HAL_StatusTypeDef HAL_HASHEx_SHA256_Finish (HASH_HandleTypeDef * hhash, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Return the computed digest in SHA256 mode.

Parameters

- **hhash:** HASH handle.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- The API waits for DCIS to be set then reads the computed digest.
- HAL_HASHEx_SHA256_Finish() can be used as well to retrieve the digest in HMAC SHA256 mode.

HAL_HMACEx_SHA224_Start

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA256_Start

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer, uint32_t Timeout)

Function description

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.
- **Timeout:** Timeout value.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_IT

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 28 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA256_Start_IT

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_IT (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size, uint8_t * pOutBuffer)

Function description

Initialize the HASH peripheral in HMAC SHA256 mode, next process pInBuffer then read the computed digest in interrupt mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.
- **pOutBuffer:** pointer to the computed digest. Digest size is 32 bytes.

Return values

- **HAL:** status

Notes

- Digest is available in pOutBuffer.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.

HAL_HMACEx_SHA224_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA224_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_SHA256_Start_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Start_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

Initialize the HASH peripheral in HMAC SHA224 mode then initiate the required DMA transfers to feed the key and the input buffer to the Peripheral.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (buffer to be hashed).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- If MDMAT bit is set before calling this function (multi-buffer HASH processing case), the input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted. For the processing of the last buffer of the thread, MDMAT bit must be reset and the buffer length (in bytes) doesn't have to be a multiple of 4.

HAL_HMACEx_MD5_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_MD5_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

MD5 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

MD5 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_MD5_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_MD5_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

MD5 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA1_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA1_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA1 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA1_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA1 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA1_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA1_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA1 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA224_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA224 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA224 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA224_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA224_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA224 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

HAL_HMACEx_SHA256_Step1_2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Step1_2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA256 HMAC step 1 completion and step 2 start in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 1 consists in writing the inner hash function key in the Peripheral, step 2 consists in writing the message text.
- The API carries out the HMAC step 1 then starts step 2 with the first buffer entered to the Peripheral. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA256_Step2_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Step2_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA256 HMAC step 2 in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral.
- The API carries on the HMAC step 2, applied to the buffer entered as input parameter. DCAL bit is not automatically set after the message buffer feeding, allowing other messages DMA transfers to occur.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- The input buffer size (in bytes) must be a multiple of 4 otherwise, the HASH digest computation is corrupted.

HAL_HMACEx_SHA256_Step2_3_DMA

Function name

HAL_StatusTypeDef HAL_HMACEx_SHA256_Step2_3_DMA (HASH_HandleTypeDef * hhash, uint8_t * pInBuffer, uint32_t Size)

Function description

SHA256 HMAC step 2 wrap-up and step 3 completion in multi-buffer DMA mode.

Parameters

- **hhash:** HASH handle.
- **pInBuffer:** pointer to the input buffer (message buffer).
- **Size:** length of the input buffer in bytes.

Return values

- **HAL:** status

Notes

- Step 2 consists in writing the message text in the Peripheral, step 3 consists in writing the outer hash function key.
- The API wraps up the HMAC step 2 in processing the buffer entered as input parameter (the input buffer must be the last one of the multi-buffer thread) then carries out HMAC step 3.
- Same key is used for the inner and the outer hash functions; pointer to key and key size are respectively stored in hhash->Init.pKey and hhash->Init.KeySize.
- Once the DMA transfers are finished (indicated by hhash->State set back to HAL_HASH_STATE_READY), HAL_HASHEx_SHA256_Finish() API must be called to retrieve the computed digest.

37 HAL HCD Generic Driver

37.1 HCD Firmware driver registers structures

37.1.1 __HCD_HandleTypeDef

__HCD_HandleTypeDef is defined in the stm32u5xx_hal_hcd.h

Data Fields

- ***HCD_TypeDef * Instance***
- ***HCD_InitTypeDef Init***
- ***HCD_HCTypeDef hc***
- ***HAL_LockTypeDef Lock***
- ***__IO HCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* PortEnabledCallback***
- ***void(* PortDisabledCallback***
- ***void(* HC_NotifyURBChangeCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***HCD_TypeDef* __HCD_HandleTypeDef::Instance***
  Register base address
- ***HCD_InitTypeDef __HCD_HandleTypeDef::Init***
  HCD required parameters
- ***HCD_HCTypeDef __HCD_HandleTypeDef::hc[16]***
  Host channels parameters
- ***HAL_LockTypeDef __HCD_HandleTypeDef::Lock***
  HCD peripheral status
- ***__IO HCD_StateTypeDef __HCD_HandleTypeDef::State***
  HCD communication state
- ***__IO uint32_t __HCD_HandleTypeDef::ErrorCode***
  HCD Error code
- ***void* __HCD_HandleTypeDef::pData***
  Pointer Stack Handler
- ***void(* __HCD_HandleTypeDef::SOFCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD SOF callback
- ***void(* __HCD_HandleTypeDef::ConnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Connect callback
- ***void(* __HCD_HandleTypeDef::DisconnectCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Disconnect callback
- ***void(* __HCD_HandleTypeDef::PortEnabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Enable callback
- ***void(* __HCD_HandleTypeDef::PortDisabledCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Port Disable callback
- ***void(* __HCD_HandleTypeDef::HC_NotifyURBChangeCallback)(struct __HCD_HandleTypeDef *hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)***
  USB OTG HCD Host Channel Notify URB Change callback

- ***void(* __HCD_HandleTypeDef::MspInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp Init callback
- ***void(* __HCD_HandleTypeDef::MspDeInitCallback)(struct __HCD_HandleTypeDef *hhcd)***
  USB OTG HCD Msp DeInit callback

37.2 HCD Firmware driver API description

The following section lists the various functions of the HCD library.

37.2.1 How to use this driver

1. Declare a HCD_HandleTypeDef handle structure, for example: HCD_HandleTypeDef hhcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_HCD_Init() API to initialize the HCD peripheral (Core, Host core, ...)
4. Initialize the HCD low level resources through the HAL_HCD_MspInit() API:
   a. Enable the HCD/USB Low Level interface clock using the following macros
      - __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
      - __HAL_RCC_USB_OTG_HS_CLK_ENABLE(); (For High Speed Mode)
      - __HAL_RCC_USB_OTG_HS_ULPI_CLK_ENABLE(); (For High Speed Mode)
      - __HAL_RCC_USB_CLK_ENABLE();
   b. Initialize the related GPIO clocks
   c. Configure HCD pin-out
   d. Configure HCD NVIC interrupt
5. Associate the Upper USB Host stack to the HAL HCD Driver:
   a. hhcd.pData = phost;
6. Enable HCD transmission and reception:
   a. HAL_HCD_Start();

37.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- ***HAL_HCD_Init()***
- ***HAL_HCD_HC_Init()***
- ***HAL_HCD_HC_Halt()***
- ***HAL_HCD_DeInit()***
- ***HAL_HCD_MspInit()***
- ***HAL_HCD_MspDeInit()***
- ***HAL_HCD_RegisterCallback()***
- ***HAL_HCD_UnRegisterCallback()***
- ***HAL_HCD_RegisterHC_NotifyURBChangeCallback()***
- ***HAL_HCD_UnRegisterHC_NotifyURBChangeCallback()***

37.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the USB Host Data Transfer

This section contains the following APIs:

- ***HAL_HCD_HC_SubmitRequest()***
- ***HAL_HCD_IRQHandler()***
- ***HAL_HCD_SOF_Callback()***
- ***HAL_HCD_Connect_Callback()***
- ***HAL_HCD_Disconnect_Callback()***
- ***HAL_HCD_PortEnabled_Callback()***
- ***HAL_HCD_PortDisabled_Callback()***

- *HAL_HCD_HC_NotifyURBChange_Callback()*
- *HAL_HCD_RegisterCallback()*
- *HAL_HCD_UnRegisterCallback()*
- *HAL_HCD_RegisterHC_NotifyURBChangeCallback()*
- *HAL_HCD_UnRegisterHC_NotifyURBChangeCallback()*
- *HAL_HCD_HC_SetHubInfo()*
- *HAL_HCD_HC_ClearHubInfo()*

37.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the HCD data transfers.

This section contains the following APIs:

- *HAL_HCD_Start()*
- *HAL_HCD_Stop()*
- *HAL_HCD_ResetPort()*

37.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_HCD_GetState()*
- *HAL_HCD_HC_GetURBState()*
- *HAL_HCD_HC_GetXferCount()*
- *HAL_HCD_HC_GetState()*
- *HAL_HCD_GetCurrentFrame()*
- *HAL_HCD_GetCurrentSpeed()*
- *HAL_HCD_HC_SetHubInfo()*
- *HAL_HCD_HC_ClearHubInfo()*

37.2.6 Detailed description of functions

HAL_HCD_Init

Function name

HAL_StatusTypeDef HAL_HCD_Init (HCD_HandleTypeDef * hhcd)

Function description

Initialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_DeInit

Function name

HAL_StatusTypeDef HAL_HCD_DeInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_Init

Function name

HAL_StatusTypeDef HAL_HCD_HC_Init (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t epnum, uint8_t dev_address, uint8_t speed, uint8_t ep_type, uint16_t mps)

Function description

Initialize a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **epnum:** Endpoint number. This parameter can be a value from 1 to 15
- **dev_address:** Current device address This parameter can be a value from 0 to 255
- **speed:** Current device speed. This parameter can be one of these values: HCD_DEVICE_SPEED_HIGH: High speed mode, HCD_DEVICE_SPEED_FULL: Full speed mode, HCD_DEVICE_SPEED_LOW: Low speed mode
- **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type, EP_TYPE_ISOC: Isochronous type, EP_TYPE_BULK: Bulk type, EP_TYPE_INTR: Interrupt type
- **mps:** Max Packet Size. This parameter can be a value from 0 to32K

Return values

- **HAL:** status

HAL_HCD_HC_Halt

Function name

HAL_StatusTypeDef HAL_HCD_HC_Halt (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

Halt a host channel.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15

Return values

- **HAL:** status

HAL_HCD_MspInit

Function name

void HAL_HCD_MspInit (HCD_HandleTypeDef * hhcd)

Function description

Initialize the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_MspDeInit

Function name

void HAL_HCD_MspDeInit (HCD_HandleTypeDef * hhcd)

Function description

DeInitialize the HCD MSP.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID, pHCD_CallbackTypeDef pCallback)

Function description

Register a User USB HCD Callback To be used instead of the weak predefined callback.

Parameters

- **hhcd:** USB HCD handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_HCD_SOF_CB_ID USB HCD SOF callback ID
  - HAL_HCD_CONNECT_CB_ID USB HCD Connect callback ID
  - HAL_HCD_DISCONNECT_CB_ID OTG HCD Disconnect callback ID
  - HAL_HCD_PORT_ENABLED_CB_ID USB HCD Port Enable callback ID
  - HAL_HCD_PORT_DISABLED_CB_ID USB HCD Port Disable callback ID
  - HAL_HCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_HCD_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_HCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterCallback (HCD_HandleTypeDef * hhcd, HAL_HCD_CallbackIDTypeDef CallbackID)

Function description

Unregister an USB HCD Callback USB HCD callback is redirected to the weak predefined callback.

Parameters

- **hhcd:** USB HCD handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_HCD_SOF_CB_ID USB HCD SOF callback ID
  - HAL_HCD_CONNECT_CB_ID USB HCD Connect callback ID
  - HAL_HCD_DISCONNECT_CB_ID OTG HCD Disconnect callback ID
  - HAL_HCD_PORT_ENABLED_CB_ID USB HCD Port Enabled callback ID
  - HAL_HCD_PORT_DISABLED_CB_ID USB HCD Port Disabled callback ID
  - HAL_HCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_HCD_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_HCD_RegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_RegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd, pHCD_HC_NotifyURBChangeCallbackTypeDef pCallback)

Function description

Register USB HCD Host Channel Notify URB Change Callback To be used instead of the weak HAL_HCD_HC_NotifyURBChange_Callback() predefined callback.

Parameters

- **hhcd:** HCD handle
- **pCallback:** pointer to the USB HCD Host Channel Notify URB Change Callback function

Return values

- **HAL:** status

HAL_HCD_UnRegisterHC_NotifyURBChangeCallback

Function name

HAL_StatusTypeDef HAL_HCD_UnRegisterHC_NotifyURBChangeCallback (HCD_HandleTypeDef * hhcd)

Function description

Unregister the USB HCD Host Channel Notify URB Change Callback USB HCD Host Channel Notify URB Change Callback is redirected to the weak HAL_HCD_HC_NotifyURBChange_Callback() predefined callback.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_HC_SubmitRequest

Function name

HAL_StatusTypeDef HAL_HCD_HC_SubmitRequest (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t direction, uint8_t ep_type, uint8_t token, uint8_t * pbuff, uint16_t length, uint8_t do_ping)

Function description

Submit a new URB for processing.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **direction:** Channel number. This parameter can be one of these values: 0 : Output / 1 : Input
- **ep_type:** Endpoint Type. This parameter can be one of these values: EP_TYPE_CTRL: Control type/ EP_TYPE_ISOC: Isochronous type/ EP_TYPE_BULK: Bulk type/ EP_TYPE_INTR: Interrupt type/
- **token:** Endpoint Type. This parameter can be one of these values: 0: HC_PID_SETUP / 1: HC_PID_DATA1
- **pbuff:** pointer to URB data
- **length:** Length of URB data
- **do_ping:** activate do ping protocol (for high speed only). This parameter can be one of these values: 0 : do ping inactive / 1 : do ping active

Return values

- **HAL:** status

HAL_HCD_HC_SetHubInfo

Function name

HAL_StatusTypeDef HAL_HCD_HC_SetHubInfo (HCD_HandleTypeDef * hhcd, uint8_t ch_num, uint8_t addr, uint8_t PortNbr)

Function description

Set host channel Hub information.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15
- **addr:** Hub address
- **PortNbr:** Hub port number

Return values

- **HAL:** status

HAL_HCD_HC_ClearHubInfo

Function name

HAL_StatusTypeDef HAL_HCD_HC_ClearHubInfo (HCD_HandleTypeDef * hhcd, uint8_t ch_num)

Function description

Clear host channel hub information.

Parameters

- **hhcd:** HCD handle
- **ch_num:** Channel number. This parameter can be a value from 1 to 15

Return values

- **HAL:** status

HAL_HCD_IRQHandler

Function name

void HAL_HCD_IRQHandler (HCD_HandleTypeDef * hhcd)

Function description

Handle HCD interrupt request.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_SOF_Callback

Function name

void HAL_HCD_SOF_Callback (HCD_HandleTypeDef * hhcd)

Function description

SOF callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Connect_Callback

Function name

void HAL_HCD_Connect_Callback (HCD_HandleTypeDef * hhcd)

Function description

Connection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_Disconnect_Callback

Function name

void HAL_HCD_Disconnect_Callback (HCD_HandleTypeDef * hhcd)

Function description

Disconnection Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_PortEnabled_Callback

Function name

void HAL_HCD_PortEnabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

Port Enabled Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_PortDisabled_Callback

Function name

void HAL_HCD_PortDisabled_Callback (HCD_HandleTypeDef * hhcd)

Function description

Port Disabled Event callback.

Parameters

- **hhcd:** HCD handle

Return values

- **None:**

HAL_HCD_HC_NotifyURBChange_Callback

Function name

void HAL_HCD_HC_NotifyURBChange_Callback (HCD_HandleTypeDef * hhcd, uint8_t chnum, HCD_URBStateTypeDef urb_state)

Function description

Notify URB state change callback.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15
- **urb_state:** This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL/

Return values

- **None:**

HAL_HCD_ResetPort

Function name

HAL_StatusTypeDef HAL_HCD_ResetPort (HCD_HandleTypeDef * hhcd)

Function description

Reset the host port.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Start

Function name

HAL_StatusTypeDef HAL_HCD_Start (HCD_HandleTypeDef * hhcd)

Function description

Start the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_Stop

Function name

HAL_StatusTypeDef HAL_HCD_Stop (HCD_HandleTypeDef * hhcd)

Function description

Stop the host driver.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** status

HAL_HCD_GetState

Function name

HCD_StateTypeDef HAL_HCD_GetState (HCD_HandleTypeDef const * hhcd)

Function description

Return the HCD handle state.

Parameters

- **hhcd:** HCD handle

Return values

- **HAL:** state

HAL_HCD_HC_GetURBState

Function name

HCD_URBStateTypeDef HAL_HCD_HC_GetURBState (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

Return URB state for a channel.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **URB:** state. This parameter can be one of these values: URB_IDLE/ URB_DONE/ URB_NOTREADY/ URB_NYET/ URB_ERROR/ URB_STALL

HAL_HCD_HC_GetState

Function name

HCD_HCStateTypeDef HAL_HCD_HC_GetState (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

Return the Host Channel state.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **Host:** channel state This parameter can be one of these values: HC_IDLE/ HC_XFRC/ HC_HALTED/ HC_NYET/ HC_NAK/ HC_STALL/ HC_XACTERR/ HC_BBLERR/ HC_DATATGLERR

HAL_HCD_HC_GetXferCount

Function name

uint32_t HAL_HCD_HC_GetXferCount (HCD_HandleTypeDef const * hhcd, uint8_t chnum)

Function description

Return the last host transfer size.

Parameters

- **hhcd:** HCD handle
- **chnum:** Channel number. This parameter can be a value from 1 to 15

Return values

- **last:** transfer size in byte

HAL_HCD_GetCurrentFrame

Function name

uint32_t HAL_HCD_GetCurrentFrame (HCD_HandleTypeDef * hhcd)

Function description

Return the current Host frame number.

Parameters

- **hhcd:** HCD handle

Return values

- **Current:** Host frame number

HAL_HCD_GetCurrentSpeed

Function name

uint32_t HAL_HCD_GetCurrentSpeed (HCD_HandleTypeDef * hhcd)

Function description

Return the Host enumeration speed.

Parameters

- **hhcd:** HCD handle

Return values

- **Enumeration:** speed

37.3 HCD Firmware driver defines

The following section lists the various define and macros of the module.

37.3.1 HCD

HCD

HCD Device Speed

HCD_DEVICE_SPEED_HIGH

HCD_DEVICE_SPEED_FULL

HCD_DEVICE_SPEED_LOW

HCD Error Code definition

HAL_HCD_ERROR_INVALID_CALLBACK

Invalid Callback error

HCD Exported Macros

__HAL_HCD_ENABLE

__HAL_HCD_DISABLE

__HAL_HCD_GET_FLAG

__HAL_HCD_GET_CH_FLAG

__HAL_HCD_CLEAR_FLAG

__HAL_HCD_IS_INVALID_INTERRUPT

__HAL_HCD_CLEAR_HC_INT

__HAL_HCD_MASK_HALT_HC_INT

__HAL_HCD_UNMASK_HALT_HC_INT

__HAL_HCD_MASK_ACK_HC_INT

__HAL_HCD_UNMASK_ACK_HC_INT

__HAL_HCD_SET_HC_CSPLT

__HAL_HCD_CLEAR_HC_CSPLT

__HAL_HCD_CLEAR_HC_SSPLT

HCD PHY Module

HCD_PHY_ULPI

HCD_PHY_EMBEDDED

HCD Speed

HCD_SPEED_HIGH

HCD_SPEED_FULL

HCD_SPEED_LOW

38 HAL I2C Generic Driver

38.1 I2C Firmware driver registers structures

38.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32u5xx_hal_i2c.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***

Field Documentation

- ***uint32_t I2C_InitTypeDef::Timing***
  Specifies the I2C_TIMINGR_register value. This parameter calculated by referring to I2C initialization section in Reference manual
- ***uint32_t I2C_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- ***uint32_t I2C_InitTypeDef::AddressingMode***
  Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of ***I2C_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***I2C_DUAL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t I2C_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected. This parameter can be a value of ***I2C_OWN_ADDRESS2_MASKS***
- ***uint32_t I2C_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***I2C_GENERAL_CALL_ADDRESSING_MODE***
- ***uint32_t I2C_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***I2C_NOSTRETCH_MODE***

38.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32u5xx_hal_i2c.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***I2C_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_StatusTypeDef(* XferISR***
- ***DMA_HandleTypeDef * hdmatx***

- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_I2C_StateTypeDef State*
- *__IO HAL_I2C_ModeTypeDef Mode*
- *__IO uint32_t ErrorCode*
- *__IO uint32_t AddrEventCount*
- *__IO uint32_t Devaddress*
- *__IO uint32_t Memaddress*
- *void(* MasterTxCpltCallback*
- *void(* MasterRxCpltCallback*
- *void(* SlaveTxCpltCallback*
- *void(* SlaveRxCpltCallback*
- *void(* ListenCpltCallback*
- *void(* MemTxCpltCallback*
- *void(* MemRxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* AddrCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***I2C_TypeDef* __I2C_HandleTypeDef::Instance***
  I2C registers base address
- ***I2C_InitTypeDef __I2C_HandleTypeDef::Init***
  I2C communication parameters
- ***uint8_t* __I2C_HandleTypeDef::pBuffPtr***
  Pointer to I2C transfer buffer
- ***uint16_t __I2C_HandleTypeDef::XferSize***
  I2C transfer size
- ***__IO uint16_t __I2C_HandleTypeDef::XferCount***
  I2C transfer counter
- ***__IO uint32_t __I2C_HandleTypeDef::XferOptions***
  I2C sequantial transfer options, this parameter can be a value of *I2C_XFEROPTIONS*
- ***__IO uint32_t __I2C_HandleTypeDef::PreviousState***
  I2C communication Previous state
- ***HAL_StatusTypeDef(* __I2C_HandleTypeDef::XferISR)(struct __I2C_HandleTypeDef *hi2c, uint32_t ITFlags, uint32_t ITSources)***
  I2C transfer IRQ handler function pointer
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmatx***
  I2C Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2C_HandleTypeDef::hdmarx***
  I2C Rx DMA handle parameters
- ***HAL_LockTypeDef __I2C_HandleTypeDef::Lock***
  I2C locking object
- ***__IO HAL_I2C_StateTypeDef __I2C_HandleTypeDef::State***
  I2C communication state
- ***__IO HAL_I2C_ModeTypeDef __I2C_HandleTypeDef::Mode***
  I2C communication mode
- ***__IO uint32_t __I2C_HandleTypeDef::ErrorCode***
  I2C Error code
- ***__IO uint32_t __I2C_HandleTypeDef::AddrEventCount***
  I2C Address Event counter

- ***__IO uint32_t __I2C_HandleTypeDef::Devaddress***
  I2C Target device address
- ***__IO uint32_t __I2C_HandleTypeDef::Memaddress***
  I2C Target memory address
- ***void(* __I2C_HandleTypeDef::MasterTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MasterRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Master Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::SlaveRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Slave Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ListenCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Listen Complete callback
- ***void(* __I2C_HandleTypeDef::MemTxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Tx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::MemRxCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Memory Rx Transfer completed callback
- ***void(* __I2C_HandleTypeDef::ErrorCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Error callback
- ***void(* __I2C_HandleTypeDef::AbortCpltCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Abort callback
- ***void(* __I2C_HandleTypeDef::AddrCallback)(struct __I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  I2C Slave Address Match callback
- ***void(* __I2C_HandleTypeDef::MspInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp Init callback
- ***void(* __I2C_HandleTypeDef::MspDeInitCallback)(struct __I2C_HandleTypeDef *hi2c)***
  I2C Msp DeInit callback

38.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

38.2.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implementing the HAL_I2C_MspInit() API:
   a. Enable the I2Cx interface clock
   b. I2C pins configuration
      - Enable the clock for the I2C GPIOs
      - Configure I2C pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the I2Cx interrupt priority
      - Enable the NVIC I2C IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
      - Enable the DMAx interface clock using
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx channel
      - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx channel

3. Configure the Communication Clock Timing, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call and Nostretch mode in the hi2c Init structure.
4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
- Receive in master mode an amount of data in non-blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Transmit_IT()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master or memory I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode or DMA mode IO sequential operation

Note: *These interfaces allow to manage a sequential transfer with a repeated start condition when a direction change during transfer*

- A specific option field manage the different steps of a sequential transfer

- Option field values are defined through I2C_XFEROPTIONS and are listed below:
  - I2C_FIRST_AND_LAST_FRAME: No sequential usage, functional is same as associated interfaces in no sequential mode
  - I2C_FIRST_FRAME: Sequential usage, this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition
  - I2C_FIRST_AND_NEXT_FRAME: Sequential usage (Master only), this option allow to manage a sequence with start condition, address and data to transfer without a final stop condition, an then permit a call the same master sequential interface several times (like HAL_I2C_Master_Seq_Transmit_IT() then HAL_I2C_Master_Seq_Transmit_IT() or HAL_I2C_Master_Seq_Transmit_DMA() then HAL_I2C_Master_Seq_Transmit_DMA())
  - I2C_NEXT_FRAME: Sequential usage, this option allow to manage a sequence with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and without a final stop condition in both cases
  - I2C_LAST_FRAME: Sequential usage, this option allow to manage a sequance with a restart condition, address and with new data to transfer if the direction change or manage only the new data to transfer if no direction change and with a final stop condition in both cases
  - I2C_LAST_FRAME_NO_STOP: Sequential usage (Master only), this option allow to manage a restart condition after several call of the same master sequential interface several times (link with option I2C_FIRST_AND_NEXT_FRAME). Usage can, transfer several bytes one by one using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_AND_NEXT_FRAME then I2C_NEXT_FRAME. Then usage of this option I2C_LAST_FRAME_NO_STOP at the last Transmit or Receive sequence permit to call the opposite interface Receive or Transmit without stopping the communication and so generate a restart condition.
  - I2C_OTHER_FRAME: Sequential usage (Master only), this option allow to manage a restart condition after each call of the same master sequential interface. Usage can, transfer several bytes one by one with a restart with slave address between each bytes using HAL_I2C_Master_Seq_Transmit_IT or HAL_I2C_Master_Seq_Receive_IT or HAL_I2C_Master_Seq_Transmit_DMA or HAL_I2C_Master_Seq_Receive_DMA with option I2C_FIRST_FRAME then I2C_OTHER_FRAME. Then usage of this option I2C_OTHER_AND_LAST_FRAME at the last frame to help automatic generation of STOP condition.

- Different sequential I2C interfaces are listed below:
  - Sequential transmit in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Transmit_IT() or using HAL_I2C_Master_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()
  - Sequential receive in master I2C mode an amount of data in non-blocking mode using HAL_I2C_Master_Seq_Receive_IT() or using HAL_I2C_Master_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
  - Abort a master or memory IT or DMA I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
    - End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
  - Enable/disable the Address listen mode in slave I2C mode using HAL_I2C_EnableListen_IT() HAL_I2C_DisableListen_IT()
    - When address slave I2C match, HAL_I2C_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master (Write/Read).
    - At Listen mode end HAL_I2C_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_ListenCpltCallback()
  - Sequential transmit in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Transmit_IT() or using HAL_I2C_Slave_Seq_Transmit_DMA()
    - At transmission end of current frame transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
  - Sequential receive in slave I2C mode an amount of data in non-blocking mode using HAL_I2C_Slave_Seq_Receive_IT() or using HAL_I2C_Slave_Seq_Receive_DMA()
    - At reception end of current frame transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
  - In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
  - Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

Interrupt mode IO MEM operation

- Write an amount of data in non-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

DMA mode IO operation

- Transmit in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterTxCpltCallback()

- Receive in master mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer, HAL_I2C_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MasterRxCpltCallback()
- Transmit in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer, HAL_I2C_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback()
- Receive in slave mode an amount of data in non-blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer, HAL_I2C_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()
- Abort a master or memory I2C process communication with Interrupt using HAL_I2C_Master_Abort_IT()
- End of abort process, HAL_I2C_AbortCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_AbortCpltCallback()
- Discard a slave I2C process communication using __HAL_I2C_GENERATE_NACK() macro. This action will inform Master to generate a Stop condition to discard the communication.

DMA mode IO MEM operation

- Write an amount of data in non-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()
- At Memory end of write transfer, HAL_I2C_MemTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemTxCpltCallback()
- Read an amount of data in non-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At Memory end of read transfer, HAL_I2C_MemRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_I2C_MemRxCpltCallback()
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_I2C_ErrorCallback()

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- __HAL_I2C_ENABLE: Enable the I2C peripheral
- __HAL_I2C_DISABLE: Disable the I2C peripheral
- __HAL_I2C_GENERATE_NACK: Generate a Non-Acknowledge I2C peripheral in Slave mode
- __HAL_I2C_GET_FLAG: Check whether the specified I2C flag is set or not
- __HAL_I2C_CLEAR_FLAG: Clear the specified I2C pending flag
- __HAL_I2C_ENABLE_IT: Enable the specified I2C interrupt
- __HAL_I2C_DISABLE_IT: Disable the specified I2C interrupt

Callback registration

The compilation flag USE_HAL_I2C_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2C_RegisterCallback() or HAL_I2C_RegisterAddrCallback() to register an interrupt callback.

Function HAL_I2C_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.

- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_I2C_RegisterAddrCallback().

Use function HAL_I2C_UnRegisterCallback to reset a callback to the default weak function. HAL_I2C_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- MemTxCpltCallback : callback for Memory transmission end of transfer.
- MemRxCpltCallback : callback for Memory reception end of transfer.
- ErrorCallback : callback for error detection.
- AbortCpltCallback : callback for abort completion process.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_I2C_UnRegisterAddrCallback().

By default, after the HAL_I2C_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2C_MasterTxCpltCallback(), HAL_I2C_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2C_Init()/ HAL_I2C_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2C_Init()/ HAL_I2C_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_I2C_RegisterCallback() before calling HAL_I2C_DeInit() or HAL_I2C_Init() function.

When the compilation flag USE_HAL_I2C_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the I2C HAL driver header file for more useful macros

38.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- *HAL_I2C_Init()*
- *HAL_I2C_DeInit()*
- *HAL_I2C_MspInit()*
- *HAL_I2C_MspDeInit()*
- *HAL_I2C_RegisterCallback()*
- *HAL_I2C_UnRegisterCallback()*
- *HAL_I2C_RegisterAddrCallback()*
- *HAL_I2C_UnRegisterAddrCallback()*

38.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:
   – Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   – No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   – HAL_I2C_Master_Transmit()
   – HAL_I2C_Master_Receive()
   – HAL_I2C_Slave_Transmit()
   – HAL_I2C_Slave_Receive()
   – HAL_I2C_Mem_Write()
   – HAL_I2C_Mem_Read()
   – HAL_I2C_IsDeviceReady()
3. No-Blocking mode functions with Interrupt are :
   – HAL_I2C_Master_Transmit_IT()
   – HAL_I2C_Master_Receive_IT()
   – HAL_I2C_Slave_Transmit_IT()
   – HAL_I2C_Slave_Receive_IT()
   – HAL_I2C_Mem_Write_IT()
   – HAL_I2C_Mem_Read_IT()
   – HAL_I2C_Master_Seq_Transmit_IT()
   – HAL_I2C_Master_Seq_Receive_IT()
   – HAL_I2C_Slave_Seq_Transmit_IT()
   – HAL_I2C_Slave_Seq_Receive_IT()
   – HAL_I2C_EnableListen_IT()
   – HAL_I2C_DisableListen_IT()
   – HAL_I2C_Master_Abort_IT()
4. No-Blocking mode functions with DMA are :
   – HAL_I2C_Master_Transmit_DMA()
   – HAL_I2C_Master_Receive_DMA()
   – HAL_I2C_Slave_Transmit_DMA()
   – HAL_I2C_Slave_Receive_DMA()
   – HAL_I2C_Mem_Write_DMA()
   – HAL_I2C_Mem_Read_DMA()
   – HAL_I2C_Master_Seq_Transmit_DMA()
   – HAL_I2C_Master_Seq_Receive_DMA()
   – HAL_I2C_Slave_Seq_Transmit_DMA()
   – HAL_I2C_Slave_Seq_Receive_DMA()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   – HAL_I2C_MasterTxCpltCallback()
   – HAL_I2C_MasterRxCpltCallback()
   – HAL_I2C_SlaveTxCpltCallback()
   – HAL_I2C_SlaveRxCpltCallback()
   – HAL_I2C_MemTxCpltCallback()
   – HAL_I2C_MemRxCpltCallback()
   – HAL_I2C_AddrCallback()
   – HAL_I2C_ListenCpltCallback()
   – HAL_I2C_ErrorCallback()
   – HAL_I2C_AbortCpltCallback()

This section contains the following APIs:
- *HAL_I2C_Master_Transmit()*
- *HAL_I2C_Master_Receive()*
- *HAL_I2C_Slave_Transmit()*
- *HAL_I2C_Slave_Receive()*
- *HAL_I2C_Master_Transmit_IT()*
- *HAL_I2C_Master_Receive_IT()*
- *HAL_I2C_Slave_Transmit_IT()*
- *HAL_I2C_Slave_Receive_IT()*
- *HAL_I2C_Master_Transmit_DMA()*
- *HAL_I2C_Master_Receive_DMA()*
- *HAL_I2C_Slave_Transmit_DMA()*
- *HAL_I2C_Slave_Receive_DMA()*
- *HAL_I2C_Mem_Write()*
- *HAL_I2C_Mem_Read()*
- *HAL_I2C_Mem_Write_IT()*
- *HAL_I2C_Mem_Read_IT()*
- *HAL_I2C_Mem_Write_DMA()*
- *HAL_I2C_Mem_Read_DMA()*
- *HAL_I2C_IsDeviceReady()*
- *HAL_I2C_Master_Seq_Transmit_IT()*
- *HAL_I2C_Master_Seq_Transmit_DMA()*
- *HAL_I2C_Master_Seq_Receive_IT()*
- *HAL_I2C_Master_Seq_Receive_DMA()*
- *HAL_I2C_Slave_Seq_Transmit_IT()*
- *HAL_I2C_Slave_Seq_Transmit_DMA()*
- *HAL_I2C_Slave_Seq_Receive_IT()*
- *HAL_I2C_Slave_Seq_Receive_DMA()*
- *HAL_I2C_EnableListen_IT()*
- *HAL_I2C_DisableListen_IT()*
- *HAL_I2C_Master_Abort_IT()*

38.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:
- *HAL_I2C_GetState()*
- *HAL_I2C_GetMode()*
- *HAL_I2C_GetError()*

38.2.5 Detailed description of functions

HAL_I2C_Init

Function name

HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)

Function description

Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and initialize the associated handle.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DeInit

Function name

HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C peripheral.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_MspInit

Function name

void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)

Function description

Initialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MspDeInit

Function name

void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)

Function description

DeInitialize the I2C MSP.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID, pI2C_CallbackTypeDef pCallback)

Function description

Register a User I2C Callback To be used instead of the weak predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_I2C_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  – HAL_I2C_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  – HAL_I2C_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  – HAL_I2C_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  – HAL_I2C_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  – HAL_I2C_MEM_TX_COMPLETE_CB_ID Memory Tx Transfer callback ID
  – HAL_I2C_MEM_RX_COMPLETE_CB_ID Memory Rx Transfer completed callback ID
  – HAL_I2C_ERROR_CB_ID Error callback ID
  – HAL_I2C_ABORT_CB_ID Abort callback ID
  – HAL_I2C_MSPINIT_CB_ID MspInit callback ID
  – HAL_I2C_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_I2C_RegisterCallback() may be called before HAL_I2C_Init() in HAL_I2C_STATE_RESET to register callbacks for HAL_I2C_MSPINIT_CB_ID and HAL_I2C_MSPDEINIT_CB_ID.

HAL_I2C_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterCallback (I2C_HandleTypeDef * hi2c, HAL_I2C_CallbackIDTypeDef CallbackID)

Function description

Unregister an I2C Callback I2C callback is redirected to the weak predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  – HAL_I2C_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  – HAL_I2C_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  – HAL_I2C_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  – HAL_I2C_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  – HAL_I2C_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  – HAL_I2C_MEM_TX_COMPLETE_CB_ID Memory Tx Transfer callback ID
  – HAL_I2C_MEM_RX_COMPLETE_CB_ID Memory Rx Transfer completed callback ID
  – HAL_I2C_ERROR_CB_ID Error callback ID
  – HAL_I2C_ABORT_CB_ID Abort callback ID
  – HAL_I2C_MSPINIT_CB_ID MspInit callback ID
  – HAL_I2C_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_I2C_UnRegisterCallback() may be called before HAL_I2C_Init() in HAL_I2C_STATE_RESET to un-register callbacks for HAL_I2C_MSPINIT_CB_ID and HAL_I2C_MSPDEINIT_CB_ID.

HAL_I2C_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_RegisterAddrCallback (I2C_HandleTypeDef * hi2c, pI2C_AddrCallbackTypeDef pCallback)

Function description

Register the Slave Address Match I2C Callback To be used instead of the weak HAL_I2C_AddrCallback() predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pCallback:** pointer to the Address Match Callback function

Return values

- **HAL:** status

HAL_I2C_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_I2C_UnRegisterAddrCallback (I2C_HandleTypeDef * hi2c)

Function description

UnRegister the Slave Address Match I2C Callback Info Ready I2C Callback is redirected to the weak HAL_I2C_AddrCallback() predefined callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_Master_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Master_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receives in master mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmits in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Slave_Receive

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive in slave mode an amount of data in blocking mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Write

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Write an amount of data in blocking mode to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_Mem_Read

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Read an amount of data in blocking mode from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_I2C_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Checks if target device is ready for communication.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- This function is used with Memory devices

HAL_I2C_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with Interrupt to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_IT

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Read an amount of data in non-blocking mode with Interrupt from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_EnableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** status

HAL_I2C_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_I2C_DisableListen_IT (I2C_HandleTypeDef * hi2c)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C

Return values

- **HAL:** status

HAL_I2C_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_I2C_Master_Abort_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress)

Function description

Abort a master or memory I2C IT or DMA process communication with Interrupt.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

HAL_I2C_Master_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Transmit in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Master_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function description

Receive in master mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Transmit in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Slave_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function description

Receive in slave mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Write_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Write an amount of data in non-blocking mode with DMA to a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_I2C_Mem_Read_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function description

Reads an amount of data in non-blocking mode with DMA from a specific memory address.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be read

Return values

- **HAL:** status

HAL_I2C_Master_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Master_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Master_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in master I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Transmit_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential transmit in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_Slave_Seq_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2C_Slave_Seq_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Sequential receive in slave/device I2C mode an amount of data in non-blocking mode with DMA.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of I2C Sequential Transfer Options

Return values

- **HAL:** status

Notes

- This interface allow to manage repeated start condition when a direction change during transfer

HAL_I2C_EV_IRQHandler

Function name

void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C event interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ER_IRQHandler

Function name

void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)

Function description

This function handles I2C error interrupt request.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterTxCpltCallback

Function name

void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MasterRxCpltCallback

Function name

void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Master Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveTxCpltCallback

Function name

void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_SlaveRxCpltCallback

Function name

void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AddrCallback

Function name

void HAL_I2C_AddrCallback (I2C_HandleTypeDef * hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
- **TransferDirection:** Master request Transfer Direction (Write/Read), value of I2C Transfer Direction Master Point of View
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_I2C_ListenCpltCallback

Function name

void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Listen Complete callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemTxCpltCallback

Function name

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Tx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_MemRxCpltCallback

Function name

void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

Memory Rx Transfer completed callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_ErrorCallback

Function name

void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C error callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_AbortCpltCallback

Function name

void HAL_I2C_AbortCpltCallback (I2C_HandleTypeDef * hi2c)

Function description

I2C abort callback.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **None:**

HAL_I2C_GetState

Function name

HAL_I2C_StateTypeDef HAL_I2C_GetState (const I2C_HandleTypeDef * hi2c)

Function description

Return the I2C handle state.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **HAL:** state

HAL_I2C_GetMode

Function name

HAL_I2C_ModeTypeDef HAL_I2C_GetMode (const I2C_HandleTypeDef * hi2c)

Function description

Returns the I2C Master, Slave, Memory or no mode.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

Return values

- **HAL:** mode

HAL_I2C_GetError

Function name

uint32_t HAL_I2C_GetError (const I2C_HandleTypeDef * hi2c)

Function description

Return the I2C error code.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.

Return values

- **I2C:** Error Code

38.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

38.3.1 I2C

I2C

I2C Addressing Mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

I2C Dual Addressing Mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

I2C Error Code definition

HAL_I2C_ERROR_NONE

No error

HAL_I2C_ERROR_BERR

BERR error

HAL_I2C_ERROR_ARLO

ARLO error

HAL_I2C_ERROR_AF

ACKF error

HAL_I2C_ERROR_OVR

OVR error

HAL_I2C_ERROR_DMA

DMA transfer error

HAL_I2C_ERROR_TIMEOUT

Timeout error

HAL_I2C_ERROR_SIZE

Size Management error

HAL_I2C_ERROR_DMA_PARAM

DMA Parameter Error

HAL_I2C_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_I2C_ERROR_INVALID_PARAM

Invalid Parameters error

I2C Exported Macros

__HAL_I2C_RESET_HANDLE_STATE

Description:

- Reset I2C handle state.

Parameters:

- __HANDLE__: specifies the I2C Handle.

Return value:

- None

__HAL_I2C_ENABLE_IT

Description:

- Enable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – I2C_IT_ERRI Errors interrupt enable
  – I2C_IT_TCI Transfer complete interrupt enable
  – I2C_IT_STOPI STOP detection interrupt enable
  – I2C_IT_NACKI NACK received interrupt enable
  – I2C_IT_ADDRI Address match interrupt enable
  – I2C_IT_RXI RX interrupt enable
  – I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_DISABLE_IT

Description:

- Disable the specified I2C interrupt.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  – I2C_IT_ERRI Errors interrupt enable
  – I2C_IT_TCI Transfer complete interrupt enable
  – I2C_IT_STOPI STOP detection interrupt enable
  – I2C_IT_NACKI NACK received interrupt enable
  – I2C_IT_ADDRI Address match interrupt enable
  – I2C_IT_RXI RX interrupt enable
  – I2C_IT_TXI TX interrupt enable

Return value:

- None

__HAL_I2C_GET_IT_SOURCE

Description:

- Check whether the specified I2C interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __INTERRUPT__: specifies the I2C interrupt source to check. This parameter can be one of the following values:
  – I2C_IT_ERRI Errors interrupt enable
  – I2C_IT_TCI Transfer complete interrupt enable
  – I2C_IT_STOPI STOP detection interrupt enable
  – I2C_IT_NACKI NACK received interrupt enable
  – I2C_IT_ADDRI Address match interrupt enable
  – I2C_IT_RXI RX interrupt enable
  – I2C_IT_TXI TX interrupt enable

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

I2C_FLAG_MASK

Description:

- Check whether the specified I2C flag is set or not.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_TXIS Transmit interrupt status
  - I2C_FLAG_RXNE Receive data register not empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_TC Transfer complete (master mode)
  - I2C_FLAG_TCR Transfer complete reload
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert
  - I2C_FLAG_BUSY Bus busy
  - I2C_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_I2C_GET_FLAG

__HAL_I2C_CLEAR_FLAG

Description:

- Clear the I2C pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - I2C_FLAG_TXE Transmit data register empty
  - I2C_FLAG_ADDR Address matched (slave mode)
  - I2C_FLAG_AF Acknowledge failure received flag
  - I2C_FLAG_STOPF STOP detection flag
  - I2C_FLAG_BERR Bus error
  - I2C_FLAG_ARLO Arbitration lost
  - I2C_FLAG_OVR Overrun/Underrun
  - I2C_FLAG_PECERR PEC error in reception
  - I2C_FLAG_TIMEOUT Timeout or Tlow detection flag
  - I2C_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_I2C_ENABLE

Description:
- Enable the specified I2C peripheral.

Parameters:
- __HANDLE__: specifies the I2C Handle.

Return value:
- None

__HAL_I2C_DISABLE

Description:
- Disable the specified I2C peripheral.

Parameters:
- __HANDLE__: specifies the I2C Handle.

Return value:
- None

__HAL_I2C_GENERATE_NACK

Description:
- Generate a Non-Acknowledge I2C peripheral in Slave mode.

Parameters:
- __HANDLE__: specifies the I2C Handle.

Return value:
- None

I2C Flag definition

I2C_FLAG_TXE

I2C_FLAG_TXIS

I2C_FLAG_RXNE

I2C_FLAG_ADDR

I2C_FLAG_AF

I2C_FLAG_STOPF

I2C_FLAG_TC

I2C_FLAG_TCR

I2C_FLAG_BERR

I2C_FLAG_ARLO

I2C_FLAG_OVR

I2C_FLAG_PECERR

I2C_FLAG_TIMEOUT

I2C_FLAG_ALERT

I2C_FLAG_BUSY

I2C_FLAG_DIR

I2C General Call Addressing Mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

I2C Interrupt configuration definition

I2C_IT_ERRI

I2C_IT_TCI

I2C_IT_STOPI

I2C_IT_NACKI

I2C_IT_ADDRI

I2C_IT_RXI

I2C_IT_TXI

I2C Memory Address Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

I2C No-Stretch Mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

I2C Own Address2 Masks

I2C_OA2_NOMASK

I2C_OA2_MASK01

I2C_OA2_MASK02

I2C_OA2_MASK03

I2C_OA2_MASK04

I2C_OA2_MASK05

I2C_OA2_MASK06

I2C_OA2_MASK07

I2C Reload End Mode

I2C_RELOAD_MODE

I2C_AUTOEND_MODE

I2C_SOFTEND_MODE

I2C Start or Stop Mode

I2C_NO_STARTSTOP

I2C_GENERATE_NO_START_READ

I2C_GENERATE_NO_START_WRITE

I2C_GENERATE_STOP

I2C_GENERATE_START_READ

I2C_GENERATE_START_WRITE

I2C Transfer Direction Master Point of View

I2C_DIRECTION_TRANSMIT

I2C_DIRECTION_RECEIVE

I2C Sequential Transfer Options

I2C_FIRST_FRAME

I2C_FIRST_AND_NEXT_FRAME

I2C_NEXT_FRAME

I2C_FIRST_AND_LAST_FRAME

I2C_LAST_FRAME

I2C_LAST_FRAME_NO_STOP

I2C_OTHER_FRAME

I2C_OTHER_AND_LAST_FRAME

39 HAL I2C Extension Driver

39.1 I2CEx Firmware driver registers structures

39.1.1 I2C_AutonomousModeConfTypeDef

I2C_AutonomousModeConfTypeDef is defined in the stm32u5xx_hal_i2c_ex.h

Data Fields

- ***uint32_t TriggerState***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerPolarity***

Field Documentation

- ***uint32_t I2C_AutonomousModeConfTypeDef::TriggerState***
  Specifies the trigger state. This parameter can be a value of ***I2CEx_AutonomousMode_FunctionalState***
- ***uint32_t I2C_AutonomousModeConfTypeDef::TriggerSelection***
  Specifies the autonomous mode trigger signal selection. This parameter can be a value of ***I2CEx_AutonomousMode_TriggerSelection***
- ***uint32_t I2C_AutonomousModeConfTypeDef::TriggerPolarity***
  Specifies the autonomous mode trigger signal polarity sensitivity. This parameter can be a value of ***I2CEx_AutonomousMode_TriggerPolarity***

39.2 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

39.2.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32U5xx devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus
- Configure Autonomous mode

39.2.2 How to use this driver

This driver provides functions to configure Noise Filter and Wake Up Feature

1. Configure I2C Analog noise filter using the function HAL_I2CEx_ConfigAnalogFilter()
2. Configure I2C Digital noise filter using the function HAL_I2CEx_ConfigDigitalFilter()
3. Configure the enable or disable of I2C Wake Up Mode using the functions :
   – HAL_I2CEx_EnableWakeUp()
   – HAL_I2CEx_DisableWakeUp()
4. Configure the enable or disable of fast mode plus driving capability using the functions :
   – HAL_I2CEx_ConfigFastModePlus()
5. Set or get or clear the autonomous mode configuration using these functions :
   – HAL_I2CEx_SetConfigAutonomousMode()
   – HAL_I2CEx_GetConfigAutonomousMode()
   – HAL_I2CEx_ClearConfigAutonomousMode()

39.2.3 Filter Mode Functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:
- *HAL_I2CEx_ConfigAnalogFilter()*
- *HAL_I2CEx_ConfigDigitalFilter()*

39.2.4 WakeUp Mode Functions

This section provides functions allowing to:
- Configure Wake Up Feature

This section contains the following APIs:
- *HAL_I2CEx_EnableWakeUp()*
- *HAL_I2CEx_DisableWakeUp()*

39.2.5 Fast Mode Plus Functions

This section provides functions allowing to:
- Configure Fast Mode Plus

This section contains the following APIs:
- *HAL_I2CEx_ConfigFastModePlus()*

39.2.6 Autonomous Mode functions

This section provides functions allowing to:
- Configure Autonomous Mode

This section contains the following APIs:
- *HAL_I2CEx_SetConfigAutonomousMode()*
- *HAL_I2CEx_GetConfigAutonomousMode()*
- *HAL_I2CEx_ClearConfigAutonomousMode()*

39.2.7 Detailed description of functions

HAL_I2CEx_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigAnalogFilter (I2C_HandleTypeDef * hi2c, uint32_t AnalogFilter)

Function description

Configure I2C Analog noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **AnalogFilter:** New state of the Analog filter.

Return values

- **HAL:** status

HAL_I2CEx_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigDigitalFilter (I2C_HandleTypeDef * hi2c, uint32_t DigitalFilter)

Function description

Configure I2C Digital noise filter.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_I2CEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_EnableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Enable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_I2CEx_DisableWakeUp (I2C_HandleTypeDef * hi2c)

Function description

Disable I2C wakeup from Stop mode(s).

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_ConfigFastModePlus

Function name

HAL_StatusTypeDef HAL_I2CEx_ConfigFastModePlus (I2C_HandleTypeDef * hi2c, uint32_t FastModePlus)

Function description

Configure I2C Fast Mode Plus.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **FastModePlus:** New state of the Fast Mode Plus.

Return values

- **HAL:** status

HAL_I2CEx_SetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_I2CEx_SetConfigAutonomousMode (I2C_HandleTypeDef * hi2c, const I2C_AutonomousModeConfTypeDef * sConfig)

Function description

Set Autonomous Mode configuration.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **sConfig:** Pointer to a I2C_AutonomousModeConfTypeDef structure that contains the configuration information of the autonomous mode for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_GetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_I2CEx_GetConfigAutonomousMode (const I2C_HandleTypeDef * hi2c, I2C_AutonomousModeConfTypeDef * sConfig)

Function description

Get Autonomous Mode configuration.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.
- **sConfig:** Pointer to a I2C_AutonomousModeConfTypeDef structure that contains the configuration information of the autonomous mode for the specified I2Cx peripheral.

Return values

- **HAL:** status

HAL_I2CEx_ClearConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_I2CEx_ClearConfigAutonomousMode (I2C_HandleTypeDef * hi2c)

Function description

Clear Autonomous Mode configuration.

Parameters

- **hi2c:** Pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2Cx peripheral.

Return values

- **HAL:** status

39.3 I2CEx Firmware driver defines

The following section lists the various define and macros of the module.

39.3.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Autonomous Mode State

I2C_AUTO_MODE_DISABLE

Autonomous mode disable

I2C_AUTO_MODE_ENABLE

Autonomous mode enable

Extended Autonomous Mode Trigger Polarity

I2C_TRIG_POLARITY_RISING

I2C HW Trigger signal on rising edge

I2C_TRIG_POLARITY_FALLING

I2C HW Trigger signal on falling edge

I2C Extended Autonomous Mode Trigger Selection

I2C_TRIG_GRP1

Trigger Group for I2C1, I2C2, I2C4, I2C5, I2C6 (depends on Product)

I2C_TRIG_GRP2

Trigger Group for I2C3

I2C_GRP1_GPDMA_CH0_TCF_TRG

HW Trigger signal is GPDMA_CH0_TRG

I2C_GRP1_GPDMA_CH1_TCF_TRG

HW Trigger signal is GPDMA_CH1_TRG

I2C_GRP1_GPDMA_CH2_TCF_TRG

HW Trigger signal is GPDMA_CH2_TRG

I2C_GRP1_GPDMA_CH3_TCF_TRG

HW Trigger signal is GPDMA_CH3_TRG

I2C_GRP1_EXTI5_TRG

HW Trigger signal is EXTI5_TRG

I2C_GRP1_EXTI9_TRG

HW Trigger signal is EXTI9_TRG

I2C_GRP1_LPTIM1_CH1_TRG

HW Trigger signal is LPTIM1_CH1_TRG

I2C_GRP1_LPTIM2_CH1_TRG

HW Trigger signal is LPTIM2_CH1_TRG

I2C_GRP1_COMP1_TRG

HW Trigger signal is COMP1_TRG

I2C_GRP1_COMP2_TRG
HW Trigger signal is COMP2_TRG

I2C_GRP1_RTC_ALRA_TRG
HW Trigger signal is RTC_ALRA_TRG

I2C_GRP1_RTC_WUT_TRG
HW Trigger signal is RTC_WUT_TRG

I2C_GRP2_LPDMA_CH0_TCF_TRG
HW Trigger signal is LPDMA_CH0_TRG

I2C_GRP2_LPDMA_CH1_TCF_TRG
HW Trigger signal is LPDMA_CH1_TRG

I2C_GRP2_LPDMA_CH2_TCF_TRG
HW Trigger signal is LPDMA_CH2_TRG

I2C_GRP2_LPDMA_CH3_TCF_TRG
HW Trigger signal is LPDMA_CH3_TRG

I2C_GRP2_EXTI5_TRG
HW Trigger signal is EXTI5_TRG

I2C_GRP2_EXTI8_TRG
HW Trigger signal is EXTI8_TRG

I2C_GRP2_LPTIM1_CH1_TRG
HW Trigger signal is LPTIM1_CH1_TRG

I2C_GRP2_LPTIM3_CH1_TRG
HW Trigger signal is LPTIM3_CH1_TRG

I2C_GRP2_COMP1_TRG
HW Trigger signal is COMP1_TRG

I2C_GRP2_COMP2_TRG
HW Trigger signal is COMP2_TRG

I2C_GRP2_RTC_ALRA_TRG
HW Trigger signal is RTC_ALRA_TRG

I2C_GRP2_RTC_WUT_TRG
HW Trigger signal is RTC_WUT_TRG

I2C Extended Fast Mode Plus

I2C_FASTMODEPLUS_ENABLE
Enable Fast Mode Plus

I2C_FASTMODEPLUS_DISABLE
Disable Fast Mode Plus

40 HAL ICACHE Generic Driver

40.1 ICACHE Firmware driver registers structures

40.1.1 ICACHE_RegionConfigTypeDef

ICACHE_RegionConfigTypeDef is defined in the stm32u5xx_hal_icache.h

Data Fields

- ***uint32_t BaseAddress***
- ***uint32_t RemapAddress***
- ***uint32_t Size***
- ***uint32_t TrafficRoute***
- ***uint32_t OutputBurstType***

Field Documentation

- ***uint32_t ICACHE_RegionConfigTypeDef::BaseAddress***
  Configures the Base address of Region i to be remapped
- ***uint32_t ICACHE_RegionConfigTypeDef::RemapAddress***
  Configures the Remap address of Region i to be remapped
- ***uint32_t ICACHE_RegionConfigTypeDef::Size***
  Configures the Region size. This parameter can be a value of ***ICACHE_Region_Size***
- ***uint32_t ICACHE_RegionConfigTypeDef::TrafficRoute***
  Selects the traffic route. This parameter can be a value of ***ICACHE_Traffic_Route***
- ***uint32_t ICACHE_RegionConfigTypeDef::OutputBurstType***
  Selects the output burst type. This parameter can be a value of ***ICACHE_Output_Burst_Type***

40.2 ICACHE Firmware driver API description

The following section lists the various functions of the ICACHE library.

40.2.1 ICACHE main features

The Instruction Cache (ICACHE) is introduced on C-AHB code bus of Cortex-M33 processor to improve performance when fetching instruction and data from both internal and external memories. It allows close to zero wait states performance.

- The ICACHE provides two performance counters (Hit and Miss), cache invalidate maintenance operation, error management and TrustZone security support.
- The ICACHE provides additionally the possibility to remap input address falling into up to four memory regions (used to remap aliased code in external memories to the internal Code region, for execution)
  ===

40.2.2 How to use this driver

The ICACHE HAL driver can be used as follows:

1. Optionally configure the Instruction Cache mode with HAL_ICACHE_ConfigAssociativityMode() if the default configuration does not suit the application requirements.
2. Enable and disable the Instruction Cache with respectively HAL_ICACHE_Enable() and HAL_ICACHE_Disable(). Use HAL_ICACHE_IsEnabled() to get the Instruction Cache status. To ensure a deterministic cache behavior after power on, system reset or after a call to @ref HAL_ICACHE_Disable(), the application must call @ref HAL_ICACHE_WaitForInvalidateComplete(). Indeed on power on, system reset or cache disable, an automatic cache invalidation procedure is launched and the cache is bypassed until the operation completes.

3. Initiate the cache maintenance invalidation procedure with either HAL_ICACHE_Invalidate() (blocking mode) or HAL_ICACHE_Invalidate_IT() (interrupt mode). When interrupt mode is used, the callback function HAL_ICACHE_InvalidateCompleteCallback() is called when the invalidate procedure is complete. The function HAL_ICACHE_WaitForInvalidateComplete() may be called to wait for the end of the invalidate procedure automatically initiated when disabling the Instruction Cache with HAL_ICACHE_Disable(). The cache operation is bypassed during the invalidation procedure.
4. Use the performance monitoring counters for Hit and Miss with the following functions: HAL_ICACHE_Monitor_Start(), HAL_ICACHE_Monitor_Stop(), HAL_ICACHE_Monitor_Reset(), HAL_ICACHE_Monitor_GetHitValue() and HAL_ICACHE_Monitor_GetMissValue()
5. Enable and disable up to four regions to remap input address from external memories to the internal Code region for execution with HAL_ICACHE_EnableRemapRegion() and HAL_ICACHE_DisableRemapRegion()

40.2.3 Initialization and control functions

This section provides functions allowing to initialize and control the Instruction Cache (mode, invalidate procedure, performance counters).

This section contains the following APIs:

- *HAL_ICACHE_ConfigAssociativityMode()*
- *HAL_ICACHE_DeInit()*
- *HAL_ICACHE_Enable()*
- *HAL_ICACHE_Disable()*
- *HAL_ICACHE_IsEnabled()*
- *HAL_ICACHE_Invalidate()*
- *HAL_ICACHE_Invalidate_IT()*
- *HAL_ICACHE_WaitForInvalidateComplete()*
- *HAL_ICACHE_Monitor_Start()*
- *HAL_ICACHE_Monitor_Stop()*
- *HAL_ICACHE_Monitor_Reset()*
- *HAL_ICACHE_Monitor_GetHitValue()*
- *HAL_ICACHE_Monitor_GetMissValue()*

40.2.4 IRQ and callback functions

This section provides functions allowing to handle ICACHE global interrupt and the associated callback functions.

This section contains the following APIs:

- *HAL_ICACHE_IRQHandler()*
- *HAL_ICACHE_InvalidateCompleteCallback()*
- *HAL_ICACHE_ErrorCallback()*

40.2.5 Memory remapped regions functions

This section provides functions allowing to manage the remapping of external memories to internal Code for execution.

This section contains the following APIs:

- *HAL_ICACHE_EnableRemapRegion()*
- *HAL_ICACHE_DisableRemapRegion()*

40.2.6 Detailed description of functions

HAL_ICACHE_Enable

Function name

HAL_StatusTypeDef HAL_ICACHE_Enable (void)

Function description

Enable the Instruction Cache.

Return values

- **HAL:** status (HAL_OK)

Notes

- This function always returns HAL_OK even if there is any ongoing cache operation. The Instruction Cache is bypassed until the cache operation completes.

HAL_ICACHE_Disable

Function name

HAL_StatusTypeDef HAL_ICACHE_Disable (void)

Function description

Disable the Instruction Cache.

Return values

- **HAL:** status (HAL_OK/HAL_TIMEOUT)

Notes

- This function waits for the cache being disabled but not for the end of the automatic cache invalidation procedure.

HAL_ICACHE_IsEnabled

Function name

uint32_t HAL_ICACHE_IsEnabled (void)

Function description

Check whether the Instruction Cache is enabled or not.

Return values

- **Status:** (0: disabled, 1: enabled)

HAL_ICACHE_ConfigAssociativityMode

Function name

HAL_StatusTypeDef HAL_ICACHE_ConfigAssociativityMode (uint32_t AssociativityMode)

Function description

Configure the Instruction Cache cache associativity mode selection.

Parameters

- **AssociativityMode:** Associativity mode selection This parameter can be one of the following values:
  – ICACHE_1WAY 1-way cache (direct mapped cache)
  – ICACHE_2WAYS 2-ways set associative cache (default)

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

HAL_ICACHE_DeInit

Function name

HAL_StatusTypeDef HAL_ICACHE_DeInit (void)

Function description

DeInitialize the Instruction Cache.

Return values

- **HAL:** status (HAL_OK)

HAL_ICACHE_Invalidate

Function name

HAL_StatusTypeDef HAL_ICACHE_Invalidate (void)

Function description

Invalidate the Instruction Cache.

Return values

- **HAL:** status (HAL_OK/HAL_ERROR/HAL_TIMEOUT)

Notes

- This function waits for the end of cache invalidation procedure and clears the associated BSYENDF flag.

HAL_ICACHE_Invalidate_IT

Function name

HAL_StatusTypeDef HAL_ICACHE_Invalidate_IT (void)

Function description

Invalidate the Instruction Cache with interrupt.

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

Notes

- This function launches cache invalidation and returns. User application shall resort to interrupt generation to check the end of the cache invalidation with the BSYENDF flag and the HAL_ICACHE_InvalidateCompleteCallback() callback.

HAL_ICACHE_WaitForInvalidateComplete

Function name

HAL_StatusTypeDef HAL_ICACHE_WaitForInvalidateComplete (void)

Function description

Wait for the end of the Instruction Cache invalidate procedure.

Return values

- **HAL:** status (HAL_OK/HAL_TIMEOUT)

Notes

- This function checks and clears the BSYENDF flag when set.

HAL_ICACHE_Monitor_Start

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Start (uint32_t MonitorType)

Function description

Start the Instruction Cache performance monitoring.

Parameters

- **MonitorType:** Monitoring type This parameter can be one of the following values:
  – ICACHE_MONITOR_HIT_MISS Hit & Miss monitoring
  – ICACHE_MONITOR_HIT Hit monitoring
  – ICACHE_MONITOR_MISS Miss monitoring

Return values

- **HAL:** status (HAL_OK)

HAL_ICACHE_Monitor_Stop

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Stop (uint32_t MonitorType)

Function description

Stop the Instruction Cache performance monitoring.

Parameters

- **MonitorType:** Monitoring type This parameter can be one of the following values:
  – ICACHE_MONITOR_HIT_MISS Hit & Miss monitoring
  – ICACHE_MONITOR_HIT Hit monitoring
  – ICACHE_MONITOR_MISS Miss monitoring

Return values

- **HAL:** status (HAL_OK)

Notes

- Stopping the monitoring does not reset the values.

HAL_ICACHE_Monitor_Reset

Function name

HAL_StatusTypeDef HAL_ICACHE_Monitor_Reset (uint32_t MonitorType)

Function description

Reset the Instruction Cache performance monitoring values.

Parameters

- **MonitorType:** Monitoring type This parameter can be one of the following values:
  – ICACHE_MONITOR_HIT_MISS Hit & Miss monitoring
  – ICACHE_MONITOR_HIT Hit monitoring
  – ICACHE_MONITOR_MISS Miss monitoring

Return values

- **HAL:** status (HAL_OK)

HAL_ICACHE_Monitor_GetHitValue

Function name

uint32_t HAL_ICACHE_Monitor_GetHitValue (void)

Function description

Get the Instruction Cache performance Hit monitoring value.

Return values

- **Hit:** monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_ICACHE_Monitor_GetMissValue

Function name

uint32_t HAL_ICACHE_Monitor_GetMissValue (void)

Function description

Get the Instruction Cache performance Miss monitoring value.

Return values

- **Miss:** monitoring value

Notes

- Upon reaching the 32-bit maximum value, monitor does not wrap.

HAL_ICACHE_IRQHandler

Function name

void HAL_ICACHE_IRQHandler (void)

Function description

Handle the Instruction Cache interrupt request.

Return values

- **None:**

Notes

- This function should be called under the ICACHE_IRQHandler().
- This function respectively disables the interrupt and clears the flag of any pending flag before calling the associated user callback.

HAL_ICACHE_InvalidateCompleteCallback

Function name

void HAL_ICACHE_InvalidateCompleteCallback (void)

Function description

Cache invalidation complete callback.

HAL_ICACHE_ErrorCallback

Function name

void HAL_ICACHE_ErrorCallback (void)

Function description

Error callback.

HAL_ICACHE_EnableRemapRegion

Function name

HAL_StatusTypeDef HAL_ICACHE_EnableRemapRegion (uint32_t Region, const ICACHE_RegionConfigTypeDef *const pRegionConfig)

Function description

Configure and enable a region for memory remapping.

Parameters

- **Region:** Region number This parameter can be a value of
  - Remapped Region number
- **pRegionConfig:** Pointer to structure of ICACHE region configuration parameters

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

Notes

- The Instruction Cache and the region must be disabled.

HAL_ICACHE_DisableRemapRegion

Function name

HAL_StatusTypeDef HAL_ICACHE_DisableRemapRegion (uint32_t Region)

Function description

Disable the memory remapping for a predefined region.

Parameters

- **Region:** Region number This parameter can be a value of
  - Remapped Region number

Return values

- **HAL:** status (HAL_OK/HAL_ERROR)

40.3 ICACHE Firmware driver defines

The following section lists the various define and macros of the module.

40.3.1 ICACHE

ICACHE

Flags

ICACHE_FLAG_BUSY

Busy flag

ICACHE_FLAG_BUSYEND

Busy end flag

ICACHE_FLAG_ERROR

Cache error flag

Flags and Interrupts Management

__HAL_ICACHE_ENABLE_IT

Description:

- Enable ICACHE interrupts.

Parameters:

- __INTERRUPT__: specifies the ICACHE interrupt sources to be enabled. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

__HAL_ICACHE_DISABLE_IT

Description:
- Disable ICACHE interrupts.

Parameters:
- __INTERRUPT__: specifies the ICACHE interrupt sources to be disabled. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

__HAL_ICACHE_GET_IT_SOURCE

Description:
- Check whether the specified ICACHE interrupt source is enabled or not.

Parameters:
- __INTERRUPT__: specifies the ICACHE interrupt source to check. This parameter can be any combination of the following values:
  - ICACHE_IT_BUSYEND Busy end interrupt
  - ICACHE_IT_ERROR Cache error interrupt

Return value:
- The: state of __INTERRUPT__ (0 or 1).

__HAL_ICACHE_GET_FLAG

Description:
- Check whether the selected ICACHE flag is set or not.

Parameters:
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - ICACHE_FLAG_BUSY Busy flag
  - ICACHE_FLAG_BUSYEND Busy end flag
  - ICACHE_FLAG_ERROR Cache error flag

Return value:
- The: state of __FLAG__ (0 or 1).

__HAL_ICACHE_CLEAR_FLAG

Description:
- Clear the selected ICACHE flags.

Parameters:
- __FLAG__: specifies the ICACHE flags to clear. This parameter can be any combination of the following values:
  - ICACHE_FLAG_BUSYEND Busy end flag
  - ICACHE_FLAG_ERROR Cache error flag

Interrupts

ICACHE_IT_BUSYEND

Busy end interrupt

ICACHE_IT_ERROR

Cache error interrupt

Monitor type

ICACHE_MONITOR_HIT_MISS

Hit & Miss monitoring

ICACHE_MONITOR_HIT

Hit monitoring

ICACHE_MONITOR_MISS

Miss monitoring

Remapped Output burst type

ICACHE_OUTPUT_BURST_WRAP

WRAP

ICACHE_OUTPUT_BURST_INCR

INCR

Remapped Region number

ICACHE_REGION_0

Region 0

ICACHE_REGION_1

Region 1

ICACHE_REGION_2

Region 2

ICACHE_REGION_3

Region 3

Remapped Region size

ICACHE_REGIONSIZE_2MB

Region size 2MB

ICACHE_REGIONSIZE_4MB

Region size 4MB

ICACHE_REGIONSIZE_8MB

Region size 8MB

ICACHE_REGIONSIZE_16MB

Region size 16MB

ICACHE_REGIONSIZE_32MB

Region size 32MB

ICACHE_REGIONSIZE_64MB

Region size 64MB

ICACHE_REGIONSIZE_128MB

Region size 128MB

Remapped Traffic route

ICACHE_MASTER1_PORT

Master1 port

ICACHE_MASTER2_PORT

Master2 port

Ways selection

ICACHE_1WAY

1-way cache (direct mapped cache)

ICACHE_2WAYS

2-ways set associative cache (default)

41 HAL IRDA Generic Driver

41.1 IRDA Firmware driver registers structures

41.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32u5xx_hal_irda.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint8_t Prescaler***
- ***uint16_t PowerMode***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t IRDA_InitTypeDef::BaudRate***
  This member configures the IRDA communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hirda->Init.BaudRate))) where usart_ker_ckpres is the IRDA input clock divided by a prescaler
- ***uint32_t IRDA_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***IRDAEx_Word_Length***
- ***uint32_t IRDA_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***IRDA_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t IRDA_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***IRDA_Transfer_Mode***
- ***uint8_t IRDA_InitTypeDef::Prescaler***
  Specifies the Prescaler value for dividing the UART/USART source clock to achieve low-power frequency.
  Note:
  – Prescaler value 0 is forbidden
- ***uint16_t IRDA_InitTypeDef::PowerMode***
  Specifies the IRDA power mode. This parameter can be a value of ***IRDA_Low_Power***
- ***uint32_t IRDA_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the IRDA clock source. This parameter can be a value of ***IRDA_ClockPrescaler***.

41.1.2 __IRDA_HandleTypeDef

__IRDA_HandleTypeDef is defined in the stm32u5xx_hal_irda.h

Data Fields

- ***USART_TypeDef * Instance***
- ***IRDA_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***

- ***uint16_t Mask***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef gState***
- ***__IO HAL_IRDA_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __IRDA_HandleTypeDef::Instance***
  USART registers base address
- ***IRDA_InitTypeDef __IRDA_HandleTypeDef::Init***
  IRDA communication parameters
- ***const uint8_t* __IRDA_HandleTypeDef::pTxBuffPtr***
  Pointer to IRDA Tx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::TxXferSize***
  IRDA Tx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::TxXferCount***
  IRDA Tx Transfer Counter
- ***uint8_t* __IRDA_HandleTypeDef::pRxBuffPtr***
  Pointer to IRDA Rx transfer Buffer
- ***uint16_t __IRDA_HandleTypeDef::RxXferSize***
  IRDA Rx Transfer size
- ***__IO uint16_t __IRDA_HandleTypeDef::RxXferCount***
  IRDA Rx Transfer Counter
- ***uint16_t __IRDA_HandleTypeDef::Mask***
  USART RX RDR register mask
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmatx***
  IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __IRDA_HandleTypeDef::hdmarx***
  IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef __IRDA_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::gState***
  IRDA state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO HAL_IRDA_StateTypeDef __IRDA_HandleTypeDef::RxState***
  IRDA state information related to Rx operations. This parameter can be a value of **HAL_IRDA_StateTypeDef**
- ***__IO uint32_t __IRDA_HandleTypeDef::ErrorCode***
  IRDA Error code
- ***void(* __IRDA_HandleTypeDef::TxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::TxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Tx Complete Callback

- ***void(* __IRDA_HandleTypeDef::RxHalfCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Half Complete Callback
- ***void(* __IRDA_HandleTypeDef::RxCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Rx Complete Callback
- ***void(* __IRDA_HandleTypeDef::ErrorCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Error Callback
- ***void(* __IRDA_HandleTypeDef::AbortCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortTransmitCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Transmit Complete Callback
- ***void(* __IRDA_HandleTypeDef::AbortReceiveCpltCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Abort Receive Complete Callback
- ***void(* __IRDA_HandleTypeDef::MspInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp Init callback
- ***void(* __IRDA_HandleTypeDef::MspDeInitCallback)(struct __IRDA_HandleTypeDef *hirda)***
  IRDA Msp DeInit callback

41.2 IRDA Firmware driver API description

The following section lists the various functions of the IRDA library.

41.2.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a IRDA_HandleTypeDef handle structure (eg. IRDA_HandleTypeDef hirda).
2. Initialize the IRDA low level resources by implementing the HAL_IRDA_MspInit() API in setting the associated USART or UART in IRDA mode:
   – Enable the USARTx/UARTx interface clock.
   – USARTx/UARTx pins configuration:
     ◦ Enable the clock for the USARTx/UARTx GPIOs.
     ◦ Configure these USARTx/UARTx pins (TX as alternate function pull-up, RX as alternate function Input).
   – NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
     ◦ Configure the USARTx/UARTx interrupt priority.
     ◦ Enable the NVIC USARTx/UARTx IRQ handle.
     ◦ The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
   – DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
     ◦ Declare a DMA handle structure for the Tx/Rx channel.
     ◦ Enable the DMAx interface clock.
     ◦ Configure the declared DMA handle structure with the required Tx/Rx parameters.
     ◦ Configure the DMA Tx/Rx channel.
     ◦ Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length and Parity and Mode(Receiver/Transmitter), the normal or low power mode and the clock prescaler in the hirda handle Init structure.

4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API.

Note: The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.

5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission half of transfer HAL_IRDA_TxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxHalfCpltCallback()
- At transmission end of transfer HAL_IRDA_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception half of transfer HAL_IRDA_RxHalfCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxHalfCpltCallback()
- At reception end of transfer HAL_IRDA_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback()
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback()

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether or not the specified IRDA interrupt is enabled

Note: You can refer to the IRDA HAL driver header file for more useful macros

41.2.2 Callback registration

The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_IRDA_RegisterCallback() to register a user callback. Function HAL_IRDA_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_IRDA_UnRegisterCallback() to reset a callback to the default weak function. HAL_IRDA_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- MspInitCallback : IRDA MspInit.
- MspDeInitCallback : IRDA MspDeInit.

By default, after the HAL_IRDA_Init() and when the state is HAL_IRDA_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_IRDA_Init() and HAL_IRDA_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_IRDA_Init() and HAL_IRDA_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_IRDA_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_IRDA_STATE_READY or HAL_IRDA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_IRDA_RegisterCallback() before calling HAL_IRDA_DeInit() or HAL_IRDA_Init() function.

When The compilation define USE_HAL_IRDA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

41.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx in asynchronous IRDA mode.

- For the asynchronous mode only these parameters can be configured:
  - Baud Rate
  - Word Length
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Power mode
  - Prescaler setting
  - Receiver/transmitter modes

The HAL_IRDA_Init() API follows the USART asynchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- *HAL_IRDA_Init()*

- *HAL_IRDA_DeInit()*
- *HAL_IRDA_MspInit()*
- *HAL_IRDA_MspDeInit()*
- *HAL_IRDA_RegisterCallback()*
- *HAL_IRDA_UnRegisterCallback()*

41.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
   – Blocking mode: the communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   – Non-Blocking mode: the communication is performed using Interrupts or DMA, these API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
   – HAL_IRDA_Transmit()
   – HAL_IRDA_Receive()
3. Non Blocking mode APIs with Interrupt are :
   – HAL_IRDA_Transmit_IT()
   – HAL_IRDA_Receive_IT()
   – HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are :
   – HAL_IRDA_Transmit_DMA()
   – HAL_IRDA_Receive_DMA()
   – HAL_IRDA_DMAPause()
   – HAL_IRDA_DMAResume()
   – HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non Blocking mode:
   – HAL_IRDA_TxHalfCpltCallback()
   – HAL_IRDA_TxCpltCallback()
   – HAL_IRDA_RxHalfCpltCallback()
   – HAL_IRDA_RxCpltCallback()
   – HAL_IRDA_ErrorCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   – HAL_IRDA_Abort()
   – HAL_IRDA_AbortTransmit()
   – HAL_IRDA_AbortReceive()
   – HAL_IRDA_Abort_IT()
   – HAL_IRDA_AbortTransmit_IT()
   – HAL_IRDA_AbortReceive_IT()
7. For Abort services based on interrupts (HAL_IRDA_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   – HAL_IRDA_AbortCpltCallback()
   – HAL_IRDA_AbortTransmitCpltCallback()
   – HAL_IRDA_AbortReceiveCpltCallback()

8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   – Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed. Transfer is kept ongoing on IRDA side. If user wants to abort it, Abort services should be called by user.
   – Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_IRDA_ErrorCallback() user callback is executed.

This section contains the following APIs:
- *HAL_IRDA_Transmit()*
- *HAL_IRDA_Receive()*
- *HAL_IRDA_Transmit_IT()*
- *HAL_IRDA_Receive_IT()*
- *HAL_IRDA_Transmit_DMA()*
- *HAL_IRDA_Receive_DMA()*
- *HAL_IRDA_DMAPause()*
- *HAL_IRDA_DMAResume()*
- *HAL_IRDA_DMAStop()*
- *HAL_IRDA_Abort()*
- *HAL_IRDA_AbortTransmit()*
- *HAL_IRDA_AbortReceive()*
- *HAL_IRDA_Abort_IT()*
- *HAL_IRDA_AbortTransmit_IT()*
- *HAL_IRDA_AbortReceive_IT()*
- *HAL_IRDA_IRQHandler()*
- *HAL_IRDA_TxCpltCallback()*
- *HAL_IRDA_TxHalfCpltCallback()*
- *HAL_IRDA_RxCpltCallback()*
- *HAL_IRDA_RxHalfCpltCallback()*
- *HAL_IRDA_ErrorCallback()*
- *HAL_IRDA_AbortCpltCallback()*
- *HAL_IRDA_AbortTransmitCpltCallback()*
- *HAL_IRDA_AbortReceiveCpltCallback()*

41.2.5 Peripheral State and Error functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process
- HAL_IRDA_GetState() API can be helpful to check in run-time the state of the IRDA peripheral handle.
- HAL_IRDA_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:
- *HAL_IRDA_GetState()*
- *HAL_IRDA_GetError()*

41.2.6 Detailed description of functions

HAL_IRDA_Init

Function name

HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA mode according to the specified parameters in the IRDA_InitTypeDef and initialize the associated handle.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DeInit

Function name

HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA peripheral.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_MspInit

Function name

void HAL_IRDA_MspInit (IRDA_HandleTypeDef * hirda)

Function description

Initialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_MspDeInit

Function name

void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)

Function description

DeInitialize the IRDA MSP.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_RegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID, pIRDA_CallbackTypeDef pCallback)

Function description

Register a User IRDA Callback To be used to override the weak predefined callback.

Parameters

- **hirda:** irda handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_IRDA_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  – HAL_IRDA_TX_COMPLETE_CB_ID Tx Complete Callback ID
  – HAL_IRDA_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  – HAL_IRDA_RX_COMPLETE_CB_ID Rx Complete Callback ID
  – HAL_IRDA_ERROR_CB_ID Error Callback ID
  – HAL_IRDA_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  – HAL_IRDA_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  – HAL_IRDA_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  – HAL_IRDA_MSPINIT_CB_ID MspInit Callback ID
  – HAL_IRDA_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_IRDA_RegisterCallback() may be called before HAL_IRDA_Init() in HAL_IRDA_STATE_RESET to register callbacks for HAL_IRDA_MSPINIT_CB_ID and HAL_IRDA_MSPDEINIT_CB_ID

HAL_IRDA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IRDA_UnRegisterCallback (IRDA_HandleTypeDef * hirda, HAL_IRDA_CallbackIDTypeDef CallbackID)

Function description

Unregister an IRDA callback IRDA callback is redirected to the weak predefined callback.

Parameters

- **hirda:** irda handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_IRDA_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  – HAL_IRDA_TX_COMPLETE_CB_ID Tx Complete Callback ID
  – HAL_IRDA_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  – HAL_IRDA_RX_COMPLETE_CB_ID Rx Complete Callback ID
  – HAL_IRDA_ERROR_CB_ID Error Callback ID
  – HAL_IRDA_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  – HAL_IRDA_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  – HAL_IRDA_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  – HAL_IRDA_MSPINIT_CB_ID MspInit Callback ID
  – HAL_IRDA_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_IRDA_UnRegisterCallback() may be called before HAL_IRDA_Init() in HAL_IRDA_STATE_RESET to un-register callbacks for HAL_IRDA_MSPINIT_CB_ID and HAL_IRDA_MSPDEINIT_CB_ID

HAL_IRDA_Transmit

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive

Function name

HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Specify timeout value.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Transmit_DMA (IRDA_HandleTypeDef * hirda, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.

HAL_IRDA_Receive_DMA

Function name

HAL_StatusTypeDef HAL_IRDA_Receive_DMA (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must reflect the number of u16 available through pData.
- When the IRDA parity is enabled (PCE = 1), the received data contains the parity bit (MSB position).

HAL_IRDA_DMAPause

Function name

HAL_StatusTypeDef HAL_IRDA_DMAPause (IRDA_HandleTypeDef * hirda)

Function description

Pause the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** status

HAL_IRDA_DMAResume

Function name

HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)

Function description

Resume the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_DMAStop

Function name

HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)

Function description

Stop the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_IRDA_Abort

Function name

HAL_StatusTypeDef HAL_IRDA_Abort (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortTransmit

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_AbortReceive

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_IRDA_Abort_IT

Function name

HAL_StatusTypeDef HAL_IRDA_Abort_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortTransmit_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_IRDA_AbortReceive_IT (IRDA_HandleTypeDef * hirda)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable IRDA Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_IRDA_IRQHandler

Function name

void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)

Function description

Handle IRDA interrupt request.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxCpltCallback

Function name

void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_RxCpltCallback

Function name

void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_TxHalfCpltCallback

Function name

void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Tx Half Transfer completed callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- **None:**

HAL_IRDA_RxHalfCpltCallback

Function name

void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

Rx Half Transfer complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_ErrorCallback

Function name

void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA error callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortCpltCallback

Function name

void HAL_IRDA_AbortCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortTransmitCpltCallback

Function name

void HAL_IRDA_AbortTransmitCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_AbortReceiveCpltCallback

Function name

void HAL_IRDA_AbortReceiveCpltCallback (IRDA_HandleTypeDef * hirda)

Function description

IRDA Abort Receive Complete callback.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **None:**

HAL_IRDA_GetState

Function name

HAL_IRDA_StateTypeDef HAL_IRDA_GetState (const IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle state.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **HAL:** state

HAL_IRDA_GetError

Function name

uint32_t HAL_IRDA_GetError (const IRDA_HandleTypeDef * hirda)

Function description

Return the IRDA handle error code.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- **IRDA:** Error Code

41.3 IRDA Firmware driver defines

The following section lists the various define and macros of the module.

41.3.1 IRDA

IRDA

IRDA Clock Prescaler

IRDA_PRESCALER_DIV1

fclk_pres = fclk

IRDA_PRESCALER_DIV2

fclk_pres = fclk/2

IRDA_PRESCALER_DIV4

fclk_pres = fclk/4

IRDA_PRESCALER_DIV6
fclk_pres = fclk/6

IRDA_PRESCALER_DIV8
fclk_pres = fclk/8

IRDA_PRESCALER_DIV10
fclk_pres = fclk/10

IRDA_PRESCALER_DIV12
fclk_pres = fclk/12

IRDA_PRESCALER_DIV16
fclk_pres = fclk/16

IRDA_PRESCALER_DIV32
fclk_pres = fclk/32

IRDA_PRESCALER_DIV64
fclk_pres = fclk/64

IRDA_PRESCALER_DIV128
fclk_pres = fclk/128

IRDA_PRESCALER_DIV256
fclk_pres = fclk/256

IRDA DMA Rx

IRDA_DMA_RX_DISABLE
IRDA DMA RX disabled

IRDA_DMA_RX_ENABLE
IRDA DMA RX enabled

IRDA DMA Tx

IRDA_DMA_TX_DISABLE
IRDA DMA TX disabled

IRDA_DMA_TX_ENABLE
IRDA DMA TX enabled

IRDA Error Code Definition

HAL_IRDA_ERROR_NONE
No error

HAL_IRDA_ERROR_PE
Parity error

HAL_IRDA_ERROR_NE
Noise error

HAL_IRDA_ERROR_FE
frame error

HAL_IRDA_ERROR_ORE

Overrun error

HAL_IRDA_ERROR_DMA

DMA transfer error

HAL_IRDA_ERROR_BUSY

Busy Error

HAL_IRDA_ERROR_INVALID_CALLBACK

Invalid Callback error

IRDA Exported Macros

__HAL_IRDA_RESET_HANDLE_STATE

Description:

- Reset IRDA handle state.

Parameters:

- __HANDLE__: IRDA handle.

Return value:

- None

__HAL_IRDA_FLUSH_DRREGISTER

Description:

- Flush the IRDA DR register.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FLAG

Description:

- Clear the specified IRDA pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - IRDA_CLEAR_PEF
  - IRDA_CLEAR_FEF
  - IRDA_CLEAR_NEF
  - IRDA_CLEAR_OREF
  - IRDA_CLEAR_TCF
  - IRDA_CLEAR_IDLEF

Return value:

- None

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:
- Clear the IRDA FE pending flag.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_CLEAR_NEFLAG

Description:
- Clear the IRDA NE pending flag.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_CLEAR_OREFLAG

Description:
- Clear the IRDA ORE pending flag.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_CLEAR_IDLEFLAG

Description:
- Clear the IRDA IDLE pending flag.

Parameters:
- __HANDLE__: specifies the IRDA Handle.

Return value:
- None

__HAL_IRDA_GET_FLAG

Description:
- Check whether the specified IRDA flag is set or not.

Parameters:
- __HANDLE__: specifies the IRDA Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - IRDA_FLAG_REACK Receive enable acknowledge flag
  - IRDA_FLAG_TEACK Transmit enable acknowledge flag
  - IRDA_FLAG_BUSY Busy flag
  - IRDA_FLAG_ABRF Auto Baud rate detection flag
  - IRDA_FLAG_ABRE Auto Baud rate detection error flag
  - IRDA_FLAG_TXE Transmit data register empty flag
  - IRDA_FLAG_TC Transmission Complete flag
  - IRDA_FLAG_RXNE Receive data register not empty flag
  - IRDA_FLAG_ORE OverRun Error flag
  - IRDA_FLAG_NE Noise Error flag
  - IRDA_FLAG_FE Framing Error flag
  - IRDA_FLAG_PE Parity Error flag

Return value:
- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_IRDA_ENABLE_IT

Description:
- Enable the specified IRDA interrupt.

Parameters:
- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:
- None

__HAL_IRDA_DISABLE_IT

Description:
- Disable the specified IRDA interrupt.

Parameters:
- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_PE Parity Error interrupt
  - IRDA_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:
- None

__HAL_IRDA_GET_IT

Description:
- Check whether the specified IRDA interrupt has occurred or not.

Parameters:
- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ORE OverRun Error interrupt
  - IRDA_IT_NE Noise Error interrupt
  - IRDA_IT_FE Framing Error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:
- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_GET_IT_SOURCE

Description:
- Check whether the specified IRDA interrupt source is enabled or not.

Parameters:
- __HANDLE__: specifies the IRDA Handle.
- __INTERRUPT__: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
  - IRDA_IT_TXE Transmit Data Register empty interrupt
  - IRDA_IT_TC Transmission complete interrupt
  - IRDA_IT_RXNE Receive Data register not empty interrupt
  - IRDA_IT_IDLE Idle line detection interrupt
  - IRDA_IT_ERR Framing, overrun or noise error interrupt
  - IRDA_IT_PE Parity Error interrupt

Return value:
- The: new state of __IT__ (SET or RESET).

__HAL_IRDA_CLEAR_IT

Description:

- Clear the specified IRDA ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  – IRDA_CLEAR_PEF Parity Error Clear Flag
  – IRDA_CLEAR_FEF Framing Error Clear Flag
  – IRDA_CLEAR_NEF Noise detected Clear Flag
  – IRDA_CLEAR_OREF OverRun Error Clear Flag
  – IRDA_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_IRDA_SEND_REQ

Description:

- Set a specific IRDA request flag.

Parameters:

- __HANDLE__: specifies the IRDA Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  – IRDA_AUTOBAUD_REQUEST Auto-Baud Rate Request
  – IRDA_RXDATA_FLUSH_REQUEST Receive Data flush Request
  – IRDA_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE

Description:

- Enable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE

Description:

- Disable the IRDA one bit sample method.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_ENABLE

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

__HAL_IRDA_DISABLE

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- __HANDLE__: specifies the IRDA Handle.

Return value:

- None

IRDA Flags

IRDA_FLAG_REACK

IRDA receive enable acknowledge flag

IRDA_FLAG_TEACK

IRDA transmit enable acknowledge flag

IRDA_FLAG_BUSY

IRDA busy flag

IRDA_FLAG_ABRF

IRDA auto Baud rate flag

IRDA_FLAG_ABRE

IRDA auto Baud rate error

IRDA_FLAG_TXE

IRDA transmit data register empty

IRDA_FLAG_TC

IRDA transmission complete

IRDA_FLAG_RXNE

IRDA read data register not empty

IRDA_FLAG_ORE

IRDA overrun error

IRDA_FLAG_NE

IRDA noise error

IRDA_FLAG_FE

IRDA frame error

IRDA_FLAG_PE

IRDA parity error

IRDA interruptions flags mask

IRDA_IT_MASK

IRDA Interruptions flags mask

IRDA_CR_MASK

IRDA control register mask

IRDA_CR_POS

IRDA control register position

IRDA_ISR_MASK

IRDA ISR register mask

IRDA_ISR_POS

IRDA ISR register position

IRDA Interrupts Definition

IRDA_IT_PE

IRDA Parity error interruption

IRDA_IT_TXE

IRDA Transmit data register empty interruption

IRDA_IT_TC

IRDA Transmission complete interruption

IRDA_IT_RXNE

IRDA Read data register not empty interruption

IRDA_IT_IDLE

IRDA Idle interruption

IRDA_IT_ERR

IRDA Error interruption

IRDA_IT_ORE

IRDA Overrun error interruption

IRDA_IT_NE

IRDA Noise error interruption

IRDA_IT_FE

IRDA Frame error interruption

IRDA Interruption Clear Flags

IRDA_CLEAR_PEF

Parity Error Clear Flag

IRDA_CLEAR_FEF

Framing Error Clear Flag

IRDA_CLEAR_NEF

Noise Error detected Clear Flag

IRDA_CLEAR_OREF

OverRun Error Clear Flag

IRDA_CLEAR_IDLEF

IDLE line detected Clear Flag

IRDA_CLEAR_TCF

Transmission Complete Clear Flag

IRDA Low Power

IRDA_POWERMODE_NORMAL

IRDA normal power mode

IRDA_POWERMODE_LOWPOWER

IRDA low power mode

IRDA Mode

IRDA_MODE_DISABLE

Associated UART disabled in IRDA mode

IRDA_MODE_ENABLE

Associated UART enabled in IRDA mode

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disabled

IRDA_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enabled

IRDA Parity

IRDA_PARITY_NONE

No parity

IRDA_PARITY_EVEN

Even parity

IRDA_PARITY_ODD

Odd parity

IRDA Request Parameters

IRDA_AUTOBAUD_REQUEST

Auto-Baud Rate Request

IRDA_RXDATA_FLUSH_REQUEST

Receive Data flush Request

IRDA_TXDATA_FLUSH_REQUEST

Transmit data flush Request

IRDA State

IRDA_STATE_DISABLE

IRDA disabled

IRDA_STATE_ENABLE

IRDA enabled

IRDA State Code Definition

HAL_IRDA_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_IRDA_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_IRDA_STATE_BUSY

An internal process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_IRDA_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_IRDA_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_IRDA_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_IRDA_STATE_ERROR

Error Value is allowed for gState only

IRDA Transfer Mode

IRDA_MODE_RX

RX mode

IRDA_MODE_TX

TX mode

IRDA_MODE_TX_RX

RX and TX mode

42 HAL IRDA Extension Driver

42.1 IRDAEx Firmware driver defines

The following section lists the various define and macros of the module.

42.1.1 IRDAEx

IRDAEx

IRDAEx Word Length

IRDA_WORDLENGTH_7B

7-bit long frame

IRDA_WORDLENGTH_8B

8-bit long frame

IRDA_WORDLENGTH_9B

9-bit long frame

43 HAL IWDG Generic Driver

43.1 IWDG Firmware driver registers structures

43.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32u5xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***
- ***uint32_t EWI***

Field Documentation

- ***uint32_t IWDG_InitTypeDef::Prescaler***
  Select the prescaler of the IWDG. This parameter can be a value of ***IWDG_Prescaler***
- ***uint32_t IWDG_InitTypeDef::Reload***
  Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::Window***
  Specifies the window value to be compared to the down-counter. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF
- ***uint32_t IWDG_InitTypeDef::EWI***
  Specifies if IWDG Early Wakeup Interrupt is enable or not and the comparator value. This parameter must be a number between Min_Data = 0 and Max_Data = 0x0FFF value 0 means that EWI is disabled

43.1.2 __IWDG_HandleTypeDef

__IWDG_HandleTypeDef is defined in the stm32u5xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***IWDG_TypeDef* __IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef __IWDG_HandleTypeDef::Init***
  IWDG required parameters
- ***void(* __IWDG_HandleTypeDef::EwiCallback)(struct __IWDG_HandleTypeDef *hiwdg)***
  IWDG Early WakeUp Interrupt callback
- ***void(* __IWDG_HandleTypeDef::MspInitCallback)(struct __IWDG_HandleTypeDef *hiwdg)***
  IWDG Msp Init callback

43.2 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

43.2.1 IWDG Generic features

- The IWDG can be started by either software or hardware (configurable through option byte).
- The IWDG is clocked by the Low-Speed Internal clock (LSI) and thus stays active even if the main clock fails.

- Once the IWDG is started, the LSI is forced ON and both cannot be disabled. The counter starts counting down from the reset value (0xFFF). When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0x0000 AAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded into the counter and the watchdog reset is prevented.
- The IWDG is implemented in the VDD voltage domain that is still functional in STOP and STANDBY mode (IWDG reset can wake up the CPU from STANDBY). IWDGRST flag in RCC_CSR register can be used to inform when an IWDG reset occurs.
- Debug mode: When the microcontroller enters debug mode (core halted), the IWDG counter either continues to work normally or stops, depending on DBG_IWDG_STOP configuration bit in DBG module, accessible through __HAL_DBGMCU_FREEZE_IWDG() and __HAL_DBGMCU_UNFREEZE_IWDG() macros.

Min-max timeout value @32KHz (LSI): ~125us / ~131.04s The IWDG timeout may vary due to LSI clock frequency dispersion. STM32U5xx devices provide the capability to measure the LSI clock frequency (LSI clock is internally connected to TIM16 CH1 input capture). The measured value can be used to have an IWDG timeout with an acceptable accuracy.

Default timeout value (necessary for IWDG_SR status register update): Constant LSI_VALUE is defined based on the nominal LSI clock frequency. This frequency being subject to variations as mentioned above, the default timeout value (defined through constant HAL_IWDG_DEFAULT_TIMEOUT below) may become too short or too long. In such cases, this default timeout value can be tuned by redefining the constant LSI_VALUE at user-application level (based, for instance, on the measured LSI clock frequency as explained above).

43.2.2 How to use this driver

1. Register callback to treat Iwdg interrupt and MspInit using HAL_IWDG_RegisterCallback().
   - Provide exiting handle as first parameter.
   - Provide which callback will be registered using one value from HAL_IWDG_CallbackIDTypeDef.
   - Provide callback function pointer.
2. Use IWDG using HAL_IWDG_Init() function to :
   - Enable instance by writing Start keyword in IWDG_KEY register. LSI clock is forced ON and IWDG counter starts counting down.
   - Enable write access to configuration registers: IWDG_PR, IWDG_RLR, IWDG_WINR, IWDG_EWCR.
   - Configure the IWDG prescaler and counter reload value. This reload value will be loaded in the IWDG counter each time the watchdog is reloaded, then the IWDG will start counting down from this value.
   - Depending on window parameter:
     - If Window Init parameter is same as Window register value, nothing more is done but reload counter value in order to exit function with exact time base.
     - Else modify Window register. This will automatically reload watchdog counter.
   - Depending on Early Wakeup Interrupt parameter:
     - If EWI is set to disable, comparator is set to 0, interrupt is disable & flag is clear.
     - Else modify EWCR register, setting comparator value, enable interrupt & clear flag.
   - Wait for status flags to be reset.
3. Then the application program must refresh the IWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_IWDG_Refresh() function.

IWDG HAL driver macros list

Below the list of most used macros in IWDG HAL driver:
- __HAL_IWDG_START: Enable the IWDG peripheral
- __HAL_IWDG_RELOAD_COUNTER: Reloads IWDG counter with value defined in the reload register

43.2.3 Initialization and Start functions

This section provides functions allowing to:
- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef of associated handle.
- Manage Window option.

- Once initialization is performed in HAL_IWDG_Init function, Watchdog is reloaded in order to exit function with correct time base.

This section contains the following APIs:
- *HAL_IWDG_Init()*
- *HAL_IWDG_MspInit()*
- *HAL_IWDG_RegisterCallback()*
- *HAL_IWDG_UnRegisterCallback()*

43.2.4 IO operation functions

This section provides functions allowing to:
- Refresh the IWDG.

This section contains the following APIs:
- *HAL_IWDG_Refresh()*
- *HAL_IWDG_IRQHandler()*
- *HAL_IWDG_EarlyWakeupCallback()*

43.2.5 Detailed description of functions

HAL_IWDG_Init

Function name

HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and start watchdog.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_MspInit

Function name

void HAL_IWDG_MspInit (IWDG_HandleTypeDef * hiwdg)

Function description

Initialize the IWDG MSP.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_IWDG_Init function is called again to change parameters.

HAL_IWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_IWDG_RegisterCallback (IWDG_HandleTypeDef * hiwdg, HAL_IWDG_CallbackIDTypeDef CallbackID, pIWDG_CallbackTypeDef pCallback)

Function description

Register a User IWDG Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hiwdg:** IWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_IWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_IWDG_MSPINIT_CB_ID MspInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_IWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_IWDG_UnRegisterCallback (IWDG_HandleTypeDef * hiwdg, HAL_IWDG_CallbackIDTypeDef CallbackID)

Function description

Unregister a IWDG Callback IWDG Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hiwdg:** IWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_IWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  - HAL_IWDG_MSPINIT_CB_ID MspInit callback ID

Return values

- **status:**

HAL_IWDG_Refresh

Function name

HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)

Function description

Refresh the IWDG.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **HAL:** status

HAL_IWDG_IRQHandler

Function name

void HAL_IWDG_IRQHandler (IWDG_HandleTypeDef * hiwdg)

Function description

Handle IWDG interrupt request.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_IWDG_Init function with EWIMode set to IWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_IWDG_EarlyWakeupCallback

Function name

void HAL_IWDG_EarlyWakeupCallback (IWDG_HandleTypeDef * hiwdg)

Function description

IWDG Early Wakeup callback.

Parameters

- **hiwdg:** pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.

Return values

- **None:**

43.3 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

43.3.1 IWDG

IWDG

IWDG Early Wakeup Interrupt Mode

IWDG_EWI_DISABLE

EWI Disable

IWDG Exported Macros

__HAL_IWDG_START

Description:

- Enable the IWDG peripheral.

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

__HAL_IWDG_RELOAD_COUNTER

Description:

- Reload IWDG counter with value defined in the reload register (write access to IWDG_PR, IWDG_RLR, IWDG_WINR, IWDG_EWCR registers disabled).

Parameters:

- __HANDLE__: IWDG handle

Return value:

- None

IWDG Prescaler

IWDG_PRESCALER_4

IWDG prescaler set to 4

IWDG_PRESCALER_8

IWDG prescaler set to 8

IWDG_PRESCALER_16

IWDG prescaler set to 16

IWDG_PRESCALER_32

IWDG prescaler set to 32

IWDG_PRESCALER_64

IWDG prescaler set to 64

IWDG_PRESCALER_128

IWDG prescaler set to 128

IWDG_PRESCALER_256

IWDG prescaler set to 256

IWDG_PRESCALER_512

IWDG prescaler set to 512

IWDG_PRESCALER_1024

IWDG prescaler set to 1024

IWDG Window option

IWDG_WINDOW_DISABLE

44 HAL JPEG Generic Driver

44.1 JPEG Firmware driver registers structures

44.1.1 JPEG_ConfTypeDef

JPEG_ConfTypeDef is defined in the stm32u5xx_hal_jpeg.h

Data Fields

- ***uint32_t ColorSpace***
- ***uint32_t ChromaSubsampling***
- ***uint32_t ImageHeight***
- ***uint32_t ImageWidth***
- ***uint32_t ImageQuality***

Field Documentation

- ***uint32_t JPEG_ConfTypeDef::ColorSpace***
  Image Color space : gray-scale, YCBCR, RGB or CMYK This parameter can be a value of ***JPEG_ColorSpace***
- ***uint32_t JPEG_ConfTypeDef::ChromaSubsampling***
  Chroma Subsampling in case of YCBCR or CMYK color space, 0-> 4:4:4 , 1-> 4:2:2, 2 -> 4:1:1, 3 -> 4:2:0 This parameter can be a value of ***JPEG_ChromaSubsampling***
- ***uint32_t JPEG_ConfTypeDef::ImageHeight***
  Image height : number of lines
- ***uint32_t JPEG_ConfTypeDef::ImageWidth***
  Image width : number of pixels per line
- ***uint32_t JPEG_ConfTypeDef::ImageQuality***
  Quality of the JPEG encoding : from 1 to 100

44.1.2 __JPEG_HandleTypeDef

__JPEG_HandleTypeDef is defined in the stm32u5xx_hal_jpeg.h

Data Fields

- ***JPEG_TypeDef * Instance***
- ***JPEG_ConfTypeDef Conf***
- ***uint8_t * pJpegInBuffPtr***
- ***uint8_t * pJpegOutBuffPtr***
- ***__IO uint32_t JpegInCount***
- ***__IO uint32_t JpegOutCount***
- ***uint32_t InDataLength***
- ***uint32_t OutDataLength***
- ***DMA_HandleTypeDef * hdmain***
- ***DMA_HandleTypeDef * hdmaout***
- ***uint8_t CustomQuanTable***
- ***uint8_t * QuantTable0***
- ***uint8_t * QuantTable1***
- ***uint8_t * QuantTable2***
- ***uint8_t * QuantTable3***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_JPEG_STATETypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint32_t Context***
- ***void(* InfoReadyCallback***
- ***void(* EncodeCpltCallback***

- *void(* DecodeCpltCallback*
- *void(* ErrorCallback*
- *void(* GetDataCallback*
- *void(* DataReadyCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***JPEG_TypeDef* __JPEG_HandleTypeDef::Instance***
  JPEG peripheral register base address
- ***JPEG_ConfTypeDef __JPEG_HandleTypeDef::Conf***
  Current JPEG encoding/decoding parameters
- ***uint8_t* __JPEG_HandleTypeDef::pJpegInBuffPtr***
  Pointer to JPEG processing (encoding, decoding,...) input buffer
- ***uint8_t* __JPEG_HandleTypeDef::pJpegOutBuffPtr***
  Pointer to JPEG processing (encoding, decoding,...) output buffer
- ***__IO uint32_t __JPEG_HandleTypeDef::JpegInCount***
  Internal Counter of input data
- ***__IO uint32_t __JPEG_HandleTypeDef::JpegOutCount***
  Internal Counter of output data
- ***uint32_t __JPEG_HandleTypeDef::InDataLength***
  Input Buffer Length in Bytes
- ***uint32_t __JPEG_HandleTypeDef::OutDataLength***
  Output Buffer Length in Bytes
- ***DMA_HandleTypeDef* __JPEG_HandleTypeDef::hdmain***
  JPEG In DMA handle parameters
- ***DMA_HandleTypeDef* __JPEG_HandleTypeDef::hdmaout***
  JPEG Out DMA handle parameters
- ***uint8_t __JPEG_HandleTypeDef::CustomQuanTable***
  If set to 1 specify that user customized quantization tables are used
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable0***
  Basic Quantization Table for component 0
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable1***
  Basic Quantization Table for component 1
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable2***
  Basic Quantization Table for component 2
- ***uint8_t* __JPEG_HandleTypeDef::QuantTable3***
  Basic Quantization Table for component 3
- ***HAL_LockTypeDef __JPEG_HandleTypeDef::Lock***
  JPEG locking object
- ***__IO HAL_JPEG_STATETypeDef __JPEG_HandleTypeDef::State***
  JPEG peripheral state
- ***__IO uint32_t __JPEG_HandleTypeDef::ErrorCode***
  JPEG Error code
- ***__IO uint32_t __JPEG_HandleTypeDef::Context***
  JPEG Internal context
- ***void(* __JPEG_HandleTypeDef::InfoReadyCallback)(struct __JPEG_HandleTypeDef *hjpeg, JPEG_ConfTypeDef *pInfo)***
  JPEG Info ready callback
- ***void(* __JPEG_HandleTypeDef::EncodeCpltCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Encode complete callback
- ***void(* __JPEG_HandleTypeDef::DecodeCpltCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Decode complete callback
- ***void(* __JPEG_HandleTypeDef::ErrorCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Error callback

- ***void(* __JPEG_HandleTypeDef::GetDataCallback)(struct __JPEG_HandleTypeDef *hjpeg, uint32_t NbDecodedData)***
  JPEG Get Data callback
- ***void(* __JPEG_HandleTypeDef::DataReadyCallback)(struct __JPEG_HandleTypeDef *hjpeg, uint8_t *pDataOut, uint32_t OutDataLength)***
  JPEG Data ready callback
- ***void(* __JPEG_HandleTypeDef::MspInitCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Msp Init callback
- ***void(* __JPEG_HandleTypeDef::MspDeInitCallback)(struct __JPEG_HandleTypeDef *hjpeg)***
  JPEG Msp DeInit callback

44.2 JPEG Firmware driver API description

The following section lists the various functions of the JPEG library.

44.2.1 How to use this driver

1. Initialize the JPEG peripheral using HAL_JPEG_Init : No initialization parameters are required. Only the call to HAL_JPEG_Init is necessary to initialize the JPEG peripheral.
2. If operation is JPEG encoding use function HAL_JPEG_ConfigEncoding to set the encoding parameters (mandatory before calling the encoding function). the application can change the encoding parameter ImageQuality from 1 to 100 to obtain a more or less quality (visual quality vs the original row image), and inversely more or less jpg file size.
3. Note that for decoding operation the JPEG peripheral output data are organized in YCbCr blocks called MCU (Minimum Coded Unit) as defioned in the JPEG specification ISO/IEC 10918-1 standard. It is up to the application to transform these YCbCr blocks to RGB data that can be display. Respectively, for Encoding operation the JPEG peripheral input should be organized in YCbCr MCU blocks. It is up to the application to perform the necessary RGB to YCbCr MCU blocks transformation before feeding the JPEG peripheral with data.
4. Use functions HAL_JPEG_Encode and HAL_JPEG_Decode to start respectively a JPEG encoding/decoding operation in polling method (blocking).
5. Use functions HAL_JPEG_Encode_IT and HAL_JPEG_Decode_IT to start respectively a JPEG encoding/decoding operation with Interrupt method (not blocking).
6. Use functions HAL_JPEG_Encode_DMA and HAL_JPEG_Decode_DMA to start respectively a JPEG encoding/decoding operation with DMA method (not blocking).
7. Callback HAL_JPEG_InfoReadyCallback is asserted if the current operation is a JPEG decoding to provide the application with JPEG image parameters. This callback is asserted when the JPEG peripheral successfully parse the JPEG header.

8. Callback HAL_JPEG_GetDataCallback is asserted for both encoding and decoding operations to inform the application that the input buffer has been consumed by the peripheral and to ask for a new data chunk if the operation (encoding/decoding) has not been complete yet.
   – This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigInputBuffer if new input data are available, or call HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_INPUT to inform the JPEG HAL driver that the ongoing operation shall pause waiting for the application to provide a new input data chunk. Once the application succeed getting new data and if the input has been paused, the application can call the function HAL_JPEG_ConfigInputBuffer to set the new input buffer and size, then resume the JPEG HAL input by calling new function HAL_JPEG_Resume. If the application has ended feeding the HAL JPEG with input data (no more input data), the application Should call the function HAL_JPEG_ConfigInputBuffer (within the callback HAL_JPEG_GetDataCallback) with the parameter InDataLength set to zero.
   – The mechanism of HAL_JPEG_ConfigInputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows to the application to provide the input data (for encoding or decoding) by chunks. If the new input data chunk is not available (because data should be read from an input file for example) the application can pause the JPEG input (using function HAL_JPEG_Pause) Once the new input data chunk is available (read from a file for example), the application can call the function HAL_JPEG_ConfigInputBuffer to provide the HAL with the new chunk then resume the JPEG HAL input by calling function HAL_JPEG_Resume.
   – The application can call functions HAL_JPEG_ConfigInputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_GetDataCallback) Once the new input chunk data available. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_GetDataCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer to the JPEG peripheral.
9. Callback HAL_JPEG_DataReadyCallback is asserted when the HAL JPEG driver has filled the given output buffer with the given size.
   – This CallBack should be implemented in the application side. It should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output buffer location and size to be used to store next data chunk. if the application is not ready to provide the output chunk location then it can call the function HAL_JPEG_Pause with parameter XferSelection set to JPEG_PAUSE_RESUME_OUTPUT to inform the JPEG HAL driver that it shall pause output data. Once the application is ready to receive the new data chunk (output buffer location free or available) it should call the function HAL_JPEG_ConfigOutputBuffer to provide the HAL JPEG driver with the new output chunk buffer location and size, then call HAL_JPEG_Resume to inform the HAL that it shall resume outputting data in the given output buffer.
   – The mechanism of HAL_JPEG_ConfigOutputBuffer/HAL_JPEG_Pause/HAL_JPEG_Resume allows the application to receive data from the JPEG peripheral by chunks. when a chunk is received, the application can pause the HAL JPEG output data to be able to process these received data (YCbCr to RGB conversion in case of decoding or data storage in case of encoding).
   – The application can call functions HAL_JPEG_ ConfigOutputBuffer then HAL_JPEG_Resume. any time (outside the HAL_JPEG_DataReadyCallback) Once the output data buffer is free to use. However, to keep data coherency, the function HAL_JPEG_Pause must be imperatively called (if necessary) within the callback HAL_JPEG_ DataReadyCallback, i.e when the HAL JPEG has ended Transferring the previous chunk buffer from the JPEG peripheral to the application.
10. Callback HAL_JPEG_EncodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG encoding operation, and all output data has been transmitted to the application.
11. Callback HAL_JPEG_DecodeCpltCallback is asserted when the HAL JPEG driver has ended the current JPEG decoding operation. and all output data has been transmitted to the application.
12. Callback HAL_JPEG_ErrorCallback is asserted when an error occurred during the current operation. the application can call the function HAL_JPEG_GetError() to retrieve the error codes.
13. By default the HAL JPEG driver uses the default quantization tables as provide in the JPEG specification (ISO/IEC 10918-1 standard) for encoding. User can change these default tables if necessary using the function HAL_JPEG_SetUserQuantTables Note that for decoding the quantization tables are automatically extracted from the JPEG header.
14. To control JPEG state you can use the following function: HAL_JPEG_GetState()

JPEG HAL driver macros list

Below the list of most used macros in JPEG HAL driver.

- __HAL_JPEG_RESET_HANDLE_STATE : Reset JPEG handle state.
- __HAL_JPEG_ENABLE : Enable the JPEG peripheral.
- __HAL_JPEG_DISABLE : Disable the JPEG peripheral.
- __HAL_JPEG_GET_FLAG : Check the specified JPEG status flag.
- __HAL_JPEG_CLEAR_FLAG : Clear the specified JPEG status flag.
- __HAL_JPEG_ENABLE_IT : Enable the specified JPEG Interrupt.
- __HAL_JPEG_DISABLE_IT : Disable the specified JPEG Interrupt.
- __HAL_JPEG_GET_IT_SOURCE : returns the state of the specified JPEG Interrupt (Enabled or disabled).

Callback registration

44.2.2 Initialization and de-initialization functions

This section provides functions allowing to:
- Initialize the JPEG peripheral and creates the associated handle
- DeInitialize the JPEG peripheral

This section contains the following APIs:
- ***HAL_JPEG_Init()***
- ***HAL_JPEG_DeInit()***
- ***HAL_JPEG_MspInit()***
- ***HAL_JPEG_MspDeInit()***
- ***HAL_JPEG_RegisterCallback()***
- ***HAL_JPEG_UnRegisterCallback()***
- ***HAL_JPEG_RegisterInfoReadyCallback()***
- ***HAL_JPEG_UnRegisterInfoReadyCallback()***
- ***HAL_JPEG_RegisterGetDataCallback()***
- ***HAL_JPEG_UnRegisterGetDataCallback()***
- ***HAL_JPEG_RegisterDataReadyCallback()***
- ***HAL_JPEG_UnRegisterDataReadyCallback()***

44.2.3 Configuration functions

This section provides functions allowing to:
- HAL_JPEG_ConfigEncoding() : JPEG encoding configuration
- HAL_JPEG_GetInfo() : Extract the image configuration from the JPEG header during the decoding
- HAL_JPEG_EnableHeaderParsing() : Enable JPEG Header parsing for decoding
- HAL_JPEG_DisableHeaderParsing() : Disable JPEG Header parsing for decoding
- HAL_JPEG_SetUserQuantTables : Modify the default Quantization tables used for JPEG encoding.

This section contains the following APIs:
- ***HAL_JPEG_ConfigEncoding()***
- ***HAL_JPEG_GetInfo()***
- ***HAL_JPEG_EnableHeaderParsing()***
- ***HAL_JPEG_DisableHeaderParsing()***
- ***HAL_JPEG_SetUserQuantTables()***

44.2.4 JPEG processing functions

This section provides functions allowing to:
- HAL_JPEG_Encode() : JPEG encoding with polling process
- HAL_JPEG_Decode() : JPEG decoding with polling process
- HAL_JPEG_Encode_IT() : JPEG encoding with interrupt process
- HAL_JPEG_Decode_IT() : JPEG decoding with interrupt process
- HAL_JPEG_Encode_DMA() : JPEG encoding with DMA process

- HAL_JPEG_Decode_DMA() : JPEG decoding with DMA process
- HAL_JPEG_Pause() : Pause the Input/Output processing
- HAL_JPEG_Resume() : Resume the JPEG Input/Output processing
- HAL_JPEG_ConfigInputBuffer() : Config Encoding/Decoding Input Buffer
- HAL_JPEG_ConfigOutputBuffer() : Config Encoding/Decoding Output Buffer
- HAL_JPEG_Abort() : Aborts the JPEG Encoding/Decoding

This section contains the following APIs:

- *HAL_JPEG_Encode()*
- *HAL_JPEG_Decode()*
- *HAL_JPEG_Encode_IT()*
- *HAL_JPEG_Decode_IT()*
- *HAL_JPEG_Encode_DMA()*
- *HAL_JPEG_Decode_DMA()*
- *HAL_JPEG_Pause()*
- *HAL_JPEG_Resume()*
- *HAL_JPEG_ConfigInputBuffer()*
- *HAL_JPEG_ConfigOutputBuffer()*
- *HAL_JPEG_Abort()*

44.2.5 JPEG Decode and Encode callback functions

This section provides callback functions:

- HAL_JPEG_InfoReadyCallback() : Decoding JPEG Info ready callback
- HAL_JPEG_EncodeCpltCallback() : Encoding complete callback.
- HAL_JPEG_DecodeCpltCallback() : Decoding complete callback.
- HAL_JPEG_ErrorCallback() : JPEG error callback.
- HAL_JPEG_GetDataCallback() : Get New Data chunk callback.
- HAL_JPEG_DataReadyCallback() : Decoded/Encoded Data ready callback.

This section contains the following APIs:

- *HAL_JPEG_InfoReadyCallback()*
- *HAL_JPEG_EncodeCpltCallback()*
- *HAL_JPEG_DecodeCpltCallback()*
- *HAL_JPEG_ErrorCallback()*
- *HAL_JPEG_GetDataCallback()*
- *HAL_JPEG_DataReadyCallback()*

44.2.6 JPEG IRQ handler management

This section provides JPEG IRQ handler function.

- HAL_JPEG_IRQHandler() : handles JPEG interrupt request

This section contains the following APIs:

- *HAL_JPEG_IRQHandler()*

44.2.7 Peripheral State and Error functions

This section provides JPEG State and Errors function.

- HAL_JPEG_GetState() : permits to get in run-time the JPEG state.
- HAL_JPEG_GetError() : Returns the JPEG error code if any.

This section contains the following APIs:

- *HAL_JPEG_GetState()*
- *HAL_JPEG_GetError()*

44.2.8 Detailed description of functions

HAL_JPEG_Init

Function name

HAL_StatusTypeDef HAL_JPEG_Init (JPEG_HandleTypeDef * hjpeg)

Function description

Initializes the JPEG according to the specified parameters in the JPEG_InitTypeDef and creates the associated handle.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_DeInit

Function name

HAL_StatusTypeDef HAL_JPEG_DeInit (JPEG_HandleTypeDef * hjpeg)

Function description

DeInitializes the JPEG peripheral.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_MspInit

Function name

void HAL_JPEG_MspInit (JPEG_HandleTypeDef * hjpeg)

Function description

Initializes the JPEG MSP.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_MspDeInit

Function name

void HAL_JPEG_MspDeInit (JPEG_HandleTypeDef * hjpeg)

Function description

DeInitializes JPEG MSP.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterCallback (JPEG_HandleTypeDef * hjpeg, HAL_JPEG_CallbackIDTypeDef CallbackID, pJPEG_CallbackTypeDef pCallback)

Function description

Register a User JPEG Callback To be used to override the weak predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_JPEG_ENCODE_CPLT_CB_ID Encode Complete callback ID
  – HAL_JPEG_DECODE_CPLT_CB_ID Decode Complete callback ID
  – HAL_JPEG_ERROR_CB_ID Error callback ID
  – HAL_JPEG_MSPINIT_CB_ID MspInit callback ID
  – HAL_JPEG_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterCallback (JPEG_HandleTypeDef * hjpeg, HAL_JPEG_CallbackIDTypeDef CallbackID)

Function description

Unregister a JPEG Callback JPEG callback is redirected to the weak predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_JPEG_ENCODE_CPLT_CB_ID Encode Complete callback ID
  – HAL_JPEG_DECODE_CPLT_CB_ID Decode Complete callback ID
  – HAL_JPEG_ERROR_CB_ID Error callback ID
  – HAL_JPEG_MSPINIT_CB_ID MspInit callback ID
  – HAL_JPEG_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_JPEG_RegisterInfoReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterInfoReadyCallback (JPEG_HandleTypeDef * hjpeg, pJPEG_InfoReadyCallbackTypeDef pCallback)

Function description

Register Info Ready JPEG Callback To be used instead of the weak HAL_JPEG_InfoReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **pCallback:** pointer to the Info Ready Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterInfoReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterInfoReadyCallback (JPEG_HandleTypeDef * hjpeg)

Function description

UnRegister the Info Ready JPEG Callback Info Ready JPEG Callback is redirected to the weak HAL_JPEG_InfoReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle

Return values

- **HAL:** status

HAL_JPEG_RegisterGetDataCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterGetDataCallback (JPEG_HandleTypeDef * hjpeg, pJPEG_GetDataCallbackTypeDef pCallback)

Function description

Register Get Data JPEG Callback To be used instead of the weak HAL_JPEG_GetDataCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **pCallback:** pointer to the Get Data Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterGetDataCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterGetDataCallback (JPEG_HandleTypeDef * hjpeg)

Function description

UnRegister the Get Data JPEG Callback Get Data JPEG Callback is redirected to the weak HAL_JPEG_GetDataCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle

Return values

- **HAL:** status

HAL_JPEG_RegisterDataReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_RegisterDataReadyCallback (JPEG_HandleTypeDef * hjpeg, pJPEG_DataReadyCallbackTypeDef pCallback)

Function description

Register Data Ready JPEG Callback To be used instead of the weak HAL_JPEG_DataReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle
- **pCallback:** pointer to the Get Data Callback function

Return values

- **HAL:** status

HAL_JPEG_UnRegisterDataReadyCallback

Function name

HAL_StatusTypeDef HAL_JPEG_UnRegisterDataReadyCallback (JPEG_HandleTypeDef * hjpeg)

Function description

UnRegister the Data Ready JPEG Callback Get Data Ready Callback is redirected to the weak HAL_JPEG_DataReadyCallback() predefined callback.

Parameters

- **hjpeg:** JPEG handle

Return values

- **HAL:** status

HAL_JPEG_ConfigEncoding

Function name

HAL_StatusTypeDef HAL_JPEG_ConfigEncoding (JPEG_HandleTypeDef * hjpeg, const JPEG_ConfTypeDef * pConf)

Function description

Set the JPEG encoding configuration.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pConf:** pointer to a JPEG_ConfTypeDef structure that contains the encoding configuration

Return values

- **HAL:** status

HAL_JPEG_GetInfo

Function name

HAL_StatusTypeDef HAL_JPEG_GetInfo (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)

Function description

Extract the image configuration from the JPEG header during the decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pInfo:** pointer to a JPEG_ConfTypeDef structure that contains The JPEG decoded header information

Return values

- **HAL:** status

HAL_JPEG_EnableHeaderParsing

Function name

HAL_StatusTypeDef HAL_JPEG_EnableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

Enable JPEG Header parsing for decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the JPEG.

Return values

- **HAL:** status

HAL_JPEG_DisableHeaderParsing

Function name

HAL_StatusTypeDef HAL_JPEG_DisableHeaderParsing (JPEG_HandleTypeDef * hjpeg)

Function description

Disable JPEG Header parsing for decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the JPEG.

Return values

- **HAL:** status

HAL_JPEG_SetUserQuantTables

Function name

HAL_StatusTypeDef HAL_JPEG_SetUserQuantTables (JPEG_HandleTypeDef * hjpeg, uint8_t * QTable0, uint8_t * QTable1, uint8_t * QTable2, uint8_t * QTable3)

Function description

Modify the default Quantization tables used for JPEG encoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **QTable0:** pointer to uint8_t , define the user quantification table for color component 1. If NULL assume no need to update the table and no error return
- **QTable1:** pointer to uint8_t , define the user quantification table for color component 2. If NULL assume no need to update the table and no error return.
- **QTable2:** pointer to uint8_t , define the user quantification table for color component 3, If NULL assume no need to update the table and no error return.
- **QTable3:** pointer to uint8_t , define the user quantification table for color component 4. If NULL assume no need to update the table and no error return.

Return values

- **HAL:** status

HAL_JPEG_Encode

Function name

HAL_StatusTypeDef HAL_JPEG_Encode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength, uint32_t Timeout)

Function description

Starts JPEG encoding with polling processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_JPEG_Decode

Function name

HAL_StatusTypeDef HAL_JPEG_Decode (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength, uint32_t Timeout)

Function description

Starts JPEG decoding with polling processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer
- **Timeout:** Specify Timeout value

Return values

- **HAL:** status

HAL_JPEG_Encode_IT

Function name

HAL_StatusTypeDef HAL_JPEG_Encode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

Starts JPEG encoding with interrupt processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Decode_IT

Function name

HAL_StatusTypeDef HAL_JPEG_Decode_IT (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)

Function description

Starts JPEG decoding with interrupt processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Encode_DMA

Function name

HAL_StatusTypeDef HAL_JPEG_Encode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataInMCU, uint32_t InDataLength, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

Starts JPEG encoding with DMA processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataInMCU:** Pointer to the Input buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOut:** Pointer to the jpeg output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Decode_DMA

Function name

HAL_StatusTypeDef HAL_JPEG_Decode_DMA (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataIn, uint32_t InDataLength, uint8_t * pDataOutMCU, uint32_t OutDataLength)

Function description

Starts JPEG decoding with DMA processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataIn:** Pointer to the input data buffer
- **InDataLength:** size in bytes Input buffer
- **pDataOutMCU:** Pointer to the Output data buffer
- **OutDataLength:** size in bytes of the Output buffer

Return values

- **HAL:** status

HAL_JPEG_Pause

Function name

HAL_StatusTypeDef HAL_JPEG_Pause (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)

Function description

Pause the JPEG Input/Output processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **XferSelection:** This parameter can be one of the following values : JPEG_PAUSE_RESUME_INPUT : Pause Input processing JPEG_PAUSE_RESUME_OUTPUT: Pause Output processing JPEG_PAUSE_RESUME_INPUT_OUTPUT: Pause Input and Output processing

Return values

- **HAL:** status

HAL_JPEG_Resume

Function name

HAL_StatusTypeDef HAL_JPEG_Resume (JPEG_HandleTypeDef * hjpeg, uint32_t XferSelection)

Function description

Resume the JPEG Input/Output processing.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **XferSelection:** This parameter can be one of the following values : JPEG_PAUSE_RESUME_INPUT : Resume Input processing JPEG_PAUSE_RESUME_OUTPUT: Resume Output processing JPEG_PAUSE_RESUME_INPUT_OUTPUT: Resume Input and Output processing

Return values

- **HAL:** status

HAL_JPEG_ConfigInputBuffer

Function name

void HAL_JPEG_ConfigInputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewInputBuffer, uint32_t InDataLength)

Function description

Config Encoding/Decoding Input Buffer.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module.
- **pNewInputBuffer:** Pointer to the new input data buffer
- **InDataLength:** Size in bytes of the new Input data buffer

Return values

- **HAL:** status

HAL_JPEG_ConfigOutputBuffer

Function name

void HAL_JPEG_ConfigOutputBuffer (JPEG_HandleTypeDef * hjpeg, uint8_t * pNewOutputBuffer, uint32_t OutDataLength)

Function description

Config Encoding/Decoding Output Buffer.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module.
- **pNewOutputBuffer:** Pointer to the new output data buffer
- **OutDataLength:** Size in bytes of the new Output data buffer

Return values

- **HAL:** status

HAL_JPEG_Abort

Function name

HAL_StatusTypeDef HAL_JPEG_Abort (JPEG_HandleTypeDef * hjpeg)

Function description

Aborts the JPEG Encoding/Decoding.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **HAL:** status

HAL_JPEG_InfoReadyCallback

Function name

void HAL_JPEG_InfoReadyCallback (JPEG_HandleTypeDef * hjpeg, JPEG_ConfTypeDef * pInfo)

Function description

Decoding JPEG Info ready callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pInfo:** pointer to a JPEG_ConfTypeDef structure that contains The JPEG decoded header information

Return values

- **None:**

HAL_JPEG_EncodeCpltCallback

Function name

void HAL_JPEG_EncodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

Encoding complete callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_DecodeCpltCallback

Function name

void HAL_JPEG_DecodeCpltCallback (JPEG_HandleTypeDef * hjpeg)

Function description

Decoding complete callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_ErrorCallback

Function name

void HAL_JPEG_ErrorCallback (JPEG_HandleTypeDef * hjpeg)

Function description

JPEG error callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_GetDataCallback

Function name

void HAL_JPEG_GetDataCallback (JPEG_HandleTypeDef * hjpeg, uint32_t NbDecodedData)

Function description

Get New Data chunk callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **NbDecodedData:** Number of consumed data in the previous chunk in bytes

Return values

- **None:**

HAL_JPEG_DataReadyCallback

Function name

void HAL_JPEG_DataReadyCallback (JPEG_HandleTypeDef * hjpeg, uint8_t * pDataOut, uint32_t OutDataLength)

Function description

Decoded/Encoded Data ready callback.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module
- **pDataOut:** pointer to the output data buffer
- **OutDataLength:** number in bytes of data available in the specified output buffer

Return values

- **None:**

HAL_JPEG_IRQHandler

Function name

void HAL_JPEG_IRQHandler (JPEG_HandleTypeDef * hjpeg)

Function description

This function handles JPEG interrupt request.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **None:**

HAL_JPEG_GetState

Function name

HAL_JPEG_STATETypeDef HAL_JPEG_GetState (const JPEG_HandleTypeDef * hjpeg)

Function description

Returns the JPEG state.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for JPEG module

Return values

- **JPEG:** state

HAL_JPEG_GetError

Function name

uint32_t HAL_JPEG_GetError (const JPEG_HandleTypeDef * hjpeg)

Function description

Return the JPEG error code.

Parameters

- **hjpeg:** pointer to a JPEG_HandleTypeDef structure that contains the configuration information for the specified JPEG.

Return values

- **JPEG:** Error Code

44.3 JPEG Firmware driver defines

The following section lists the various define and macros of the module.

44.3.1 JPEG

JPEG

JPEG Chrominance Sampling

JPEG_444_SUBSAMPLING

Chroma Subsampling 4:4:4

JPEG_420_SUBSAMPLING

Chroma Subsampling 4:2:0

JPEG_422_SUBSAMPLING

Chroma Subsampling 4:2:2

JPEG ColorSpace

JPEG_GRAYSCALE_COLORSPACE

JPEG_YCBCR_COLORSPACE

JPEG_CMYK_COLORSPACE

JPEG Error Code definition

HAL_JPEG_ERROR_NONE

No error

HAL_JPEG_ERROR_HUFF_TABLE

HUffman Table programming error

HAL_JPEG_ERROR_QUANT_TABLE

Quantization Table programming error

HAL_JPEG_ERROR_DMA

DMA transfer error

HAL_JPEG_ERROR_TIMEOUT

Timeout error

HAL_JPEG_ERROR_INVALID_CALLBACK

Invalid Callback error

JPEG Exported Macros

__HAL_JPEG_RESET_HANDLE_STATE

Description:
- Reset JPEG handle state.

Parameters:
- __HANDLE__: specifies the JPEG handle.

Return value:
- None

__HAL_JPEG_ENABLE

Description:
- Enable the JPEG peripheral.

Parameters:
- __HANDLE__: specifies the JPEG handle.

Return value:
- None

__HAL_JPEG_DISABLE

Description:
- Disable the JPEG peripheral.

Parameters:
- __HANDLE__: specifies the JPEG handle.

Return value:
- None

__HAL_JPEG_GET_FLAG

Description:
- Check the specified JPEG status flag.

Parameters:
- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to check This parameter can be one of the following values:
  – JPEG_FLAG_IFTF : The input FIFO is not full and is below its threshold flag
  – JPEG_FLAG_IFNFF : The input FIFO Not Full Flag, a data can be written
  – JPEG_FLAG_OFTF : The output FIFO is not empty and has reach its threshold
  – JPEG_FLAG_OFNEF : The output FIFO is not empty, a data is available
  – JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
  – JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers and the internal registers have been updated
  – JPEG_FLAG_COF : JPEG Codec operation on going flag

Return value:
- __HAL_JPEG_GET_FLAG: : returns The new state of __FLAG__ (TRUE or FALSE)

__HAL_JPEG_CLEAR_FLAG

Description:

- Clear the specified JPEG status flag.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __FLAG__: specifies the flag to clear This parameter can be one of the following values:
  – JPEG_FLAG_EOCF : JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO
  – JPEG_FLAG_HPDF : JPEG Codec has finished the parsing of the headers

Return value:

- None

__HAL_JPEG_ENABLE_IT

Description:

- Enable Interrupt.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to enable This parameter can be one of the following values:
  – JPEG_IT_IFT : Input FIFO Threshold Interrupt
  – JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  – JPEG_IT_OFT : Output FIFO Threshold Interrupt
  – JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  – JPEG_IT_EOC : End of Conversion Interrupt
  – JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- No: return

__HAL_JPEG_DISABLE_IT

Description:

- Disable Interrupt.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to disable This parameter can be one of the following values:
  – JPEG_IT_IFT : Input FIFO Threshold Interrupt
  – JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  – JPEG_IT_OFT : Output FIFO Threshold Interrupt
  – JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  – JPEG_IT_EOC : End of Conversion Interrupt
  – JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- No: return

Notes:

- To disable an IT we must use MODIFY_REG macro to avoid writing "1" to the FIFO flush bits located in the same IT enable register (CR register).

__HAL_JPEG_GET_IT_SOURCE

Description:

- Get Interrupt state.

Parameters:

- __HANDLE__: specifies the JPEG handle.
- __INTERRUPT__: specifies the interrupt to check This parameter can be one of the following values:
  – JPEG_IT_IFT : Input FIFO Threshold Interrupt
  – JPEG_IT_IFNF : Input FIFO Not Full Interrupt
  – JPEG_IT_OFT : Output FIFO Threshold Interrupt
  – JPEG_IT_OFNE : Output FIFO Not empty Interrupt
  – JPEG_IT_EOC : End of Conversion Interrupt
  – JPEG_IT_HPD : Header Parsing Done Interrupt

Return value:

- returns: The new state of __INTERRUPT__ (Enabled or disabled)

JPEG Flag definition

JPEG_FLAG_IFTF

Input FIFO is not full and is below its threshold flag

JPEG_FLAG_IFNFF

Input FIFO Not Full Flag, a data can be written

JPEG_FLAG_OFTF

Output FIFO is not empty and has reach its threshold

JPEG_FLAG_OFNEF

Output FIFO is not empty, a data is available

JPEG_FLAG_EOCF

JPEG Codec core has finished the encoding or the decoding process and than last data has been sent to the output FIFO

JPEG_FLAG_HPDF

JPEG Codec has finished the parsing of the headers and the internal registers have been updated

JPEG_FLAG_COF

JPEG Codec operation on going flag

JPEG_FLAG_ALL

JPEG Codec All previous flag

JPEG Image Quality

JPEG_IMAGE_QUALITY_MIN

Minimum JPEG quality

JPEG_IMAGE_QUALITY_MAX

Maximum JPEG quality

JPEG Interrupt configuration definition

JPEG_IT_IFT

Input FIFO Threshold Interrupt

JPEG_IT_IFNF

Input FIFO Not Full Interrupt

JPEG_IT_OFT

Output FIFO Threshold Interrupt

JPEG_IT_OFNE

Output FIFO Not Empty Interrupt

JPEG_IT_EOC

End of Conversion Interrupt

JPEG_IT_HPD

Header Parsing Done Interrupt

JPEG Process Pause Resume definition

JPEG_PAUSE_RESUME_INPUT

Pause/Resume Input FIFO Xfer

JPEG_PAUSE_RESUME_OUTPUT

Pause/Resume Output FIFO Xfer

JPEG_PAUSE_RESUME_INPUT_OUTPUT

Pause/Resume Input and Output FIFO Xfer

JPEG Quantization Table Size

JPEG_QUANT_TABLE_SIZE

JPEG Quantization Table Size in bytes

45 HAL LPTIM Generic Driver

45.1 LPTIM Firmware driver registers structures

45.1.1 LPTIM_ClockConfigTypeDef

LPTIM_ClockConfigTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Prescaler***

Field Documentation

- ***uint32_t LPTIM_ClockConfigTypeDef::Source***
  Selects the clock source. This parameter can be a value of ***LPTIM_Clock_Source***
- ***uint32_t LPTIM_ClockConfigTypeDef::Prescaler***
  Specifies the counter clock Prescaler. This parameter can be a value of ***LPTIM_Clock_Prescaler***

45.1.2 LPTIM_ULPClockConfigTypeDef

LPTIM_ULPClockConfigTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_ULPClockConfigTypeDef::Polarity***
  Selects the polarity of the active edge for the counter unit if the ULPTIM input is selected. Note: This parameter is used only when Ultra low power clock source is used. Note: If the polarity is configured on 'both edges', an auxiliary clock (one of the Low power oscillator) must be active. This parameter can be a value of ***LPTIM_Clock_Polarity***
- ***uint32_t LPTIM_ULPClockConfigTypeDef::SampleTime***
  Selects the clock sampling time to configure the clock glitch filter. Note: This parameter is used only when Ultra low power clock source is used. This parameter can be a value of ***LPTIM_Clock_Sample_Time***

45.1.3 LPTIM_TriggerConfigTypeDef

LPTIM_TriggerConfigTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***uint32_t Source***
- ***uint32_t ActiveEdge***
- ***uint32_t SampleTime***

Field Documentation

- ***uint32_t LPTIM_TriggerConfigTypeDef::Source***
  Selects the Trigger source. This parameter can be a value of ***LPTIM_Trigger_Source***
- ***uint32_t LPTIM_TriggerConfigTypeDef::ActiveEdge***
  Selects the Trigger active edge. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_External_Trigger_Polarity***
- ***uint32_t LPTIM_TriggerConfigTypeDef::SampleTime***
  Selects the trigger sampling time to configure the clock glitch filter. Note: This parameter is used only when an external trigger is used. This parameter can be a value of ***LPTIM_Trigger_Sample_Time***

45.1.4 LPTIM_InitTypeDef

LPTIM_InitTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***LPTIM_ClockConfigTypeDef Clock***
- ***LPTIM_ULPClockConfigTypeDef UltraLowPowerClock***

- ***LPTIM_TriggerConfigTypeDef Trigger***
- ***uint32_t Period***
- ***uint32_t UpdateMode***
- ***uint32_t CounterSource***
- ***uint32_t Input1Source***
- ***uint32_t Input2Source***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***LPTIM_ClockConfigTypeDef LPTIM_InitTypeDef::Clock***
  Specifies the clock parameters
- ***LPTIM_ULPClockConfigTypeDef LPTIM_InitTypeDef::UltraLowPowerClock***
  Specifies the Ultra Low Power clock parameters
- ***LPTIM_TriggerConfigTypeDef LPTIM_InitTypeDef::Trigger***
  Specifies the Trigger parameters
- ***uint32_t LPTIM_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0001 and Max_Data = 0xFFFF.
- ***uint32_t LPTIM_InitTypeDef::UpdateMode***
  Specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period. This parameter can be a value of ***LPTIM_Updating_Mode***
- ***uint32_t LPTIM_InitTypeDef::CounterSource***
  Specifies whether the counter is incremented each internal event or each external event. This parameter can be a value of ***LPTIM_Counter_Source***
- ***uint32_t LPTIM_InitTypeDef::Input1Source***
  Specifies source selected for input1 (GPIO or comparator output). This parameter can be a value of ***LPTIM_Input1_Source***
- ***uint32_t LPTIM_InitTypeDef::Input2Source***
  Specifies source selected for input2 (GPIO or comparator output). Note: This parameter is used only for encoder feature so is used only for LPTIM1 instance. This parameter can be a value of ***LPTIM_Input2_Source***
- ***uint32_t LPTIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). Note: When using repetition counter the UpdateMode field must be set to LPTIM_UPDATE_ENDOFPERIOD otherwise unpredictable behavior may occur. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

45.1.5 LPTIM_OC_ConfigTypeDef

LPTIM_OC_ConfigTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***uint32_t Pulse***
- ***uint32_t OCPolarity***

Field Documentation

- ***uint32_t LPTIM_OC_ConfigTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t LPTIM_OC_ConfigTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***LPTIM_Output_Compare_Polarity***

45.1.6 LPTIM_IC_ConfigTypeDef

LPTIM_IC_ConfigTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***uint32_t ICInputSource***
- ***uint32_t ICPrescaler***
- ***uint32_t ICPolarity***

- *uint32_t ICFilter*

Field Documentation

- ***uint32_t LPTIM_IC_ConfigTypeDef::ICInputSource***
  Specifies source selected for IC channel. This parameter can be a value of *LPTIM_Input_Capture_Source*
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of *LPTIM_Input_Capture_Prescaler*
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of *LPTIM_Input_Capture_Polarity*
- ***uint32_t LPTIM_IC_ConfigTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of *LPTIM_Input_Capture_Filter*

45.1.7 __LPTIM_HandleTypeDef

__LPTIM_HandleTypeDef is defined in the stm32u5xx_hal_lptim.h

Data Fields

- ***LPTIM_TypeDef * Instance***
- ***LPTIM_InitTypeDef Init***
- ***HAL_LPTIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LPTIM_StateTypeDef State***
- ***__IO HAL_LPTIM_ChannelStateTypeDef ChannelState***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* CompareMatchCallback***
- ***void(* AutoReloadMatchCallback***
- ***void(* TriggerCallback***
- ***void(* CompareWriteCallback***
- ***void(* AutoReloadWriteCallback***
- ***void(* DirectionUpCallback***
- ***void(* DirectionDownCallback***
- ***void(* UpdateEventCallback***
- ***void(* RepCounterWriteCallback***
- ***void(* UpdateEventHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* IC_CaptureCallback***
- ***void(* IC_CaptureHalfCpltCallback***
- ***void(* IC_OverCaptureCallback***

Field Documentation

- ***LPTIM_TypeDef* __LPTIM_HandleTypeDef::Instance***
  Register base address
- ***LPTIM_InitTypeDef __LPTIM_HandleTypeDef::Init***
  LPTIM required parameters
- ***HAL_LPTIM_ActiveChannel __LPTIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __LPTIM_HandleTypeDef::hdma[3]***
  DMA Handlers array, This array is accessed by a *LPTIM_DMA_Handle_index*
- ***HAL_StatusTypeDef __LPTIM_HandleTypeDef::Status***
  LPTIM peripheral status

- ***HAL_LockTypeDef __LPTIM_HandleTypeDef::Lock***
  LPTIM locking object
- ***__IO HAL_LPTIM_StateTypeDef __LPTIM_HandleTypeDef::State***
  LPTIM peripheral state
- ***__IO HAL_LPTIM_ChannelStateTypeDef __LPTIM_HandleTypeDef::ChannelState[2]***
  LPTIM channel operation state
- ***void(* __LPTIM_HandleTypeDef::MspInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp Init Callback
- ***void(* __LPTIM_HandleTypeDef::MspDeInitCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Base Msp DeInit Callback
- ***void(* __LPTIM_HandleTypeDef::CompareMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare match Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadMatchCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload match Callback
- ***void(* __LPTIM_HandleTypeDef::TriggerCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  External trigger event detection Callback
- ***void(* __LPTIM_HandleTypeDef::CompareWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Compare register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::AutoReloadWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Auto-reload register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionUpCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Up-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::DirectionDownCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Down-counting direction change Callback
- ***void(* __LPTIM_HandleTypeDef::UpdateEventCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Update event detection Callback
- ***void(* __LPTIM_HandleTypeDef::RepCounterWriteCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Repetition counter register write complete Callback
- ***void(* __LPTIM_HandleTypeDef::UpdateEventHalfCpltCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Update event half complete detection Callback
- ***void(* __LPTIM_HandleTypeDef::ErrorCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  LPTIM Error Callback
- ***void(* __LPTIM_HandleTypeDef::IC_CaptureCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Input capture Callback
- ***void(* __LPTIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __LPTIM_HandleTypeDef *htim)***
  Input Capture half complete Callback
- ***void(* __LPTIM_HandleTypeDef::IC_OverCaptureCallback)(struct __LPTIM_HandleTypeDef *hlptim)***
  Over capture Callback

45.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

45.2.1 How to use this driver

The LPTIM HAL driver can be used as follows:

1. Initialize the LPTIM low level resources by implementing the HAL_LPTIM_MspInit():
   - Enable the LPTIM interface clock using __HAL_RCC_LPTIMx_CLK_ENABLE().
   - In case of using interrupts (e.g. HAL_LPTIM_PWM_Start_IT()):
     - Configure the LPTIM interrupt priority using HAL_NVIC_SetPriority().
     - Enable the LPTIM IRQ handler using HAL_NVIC_EnableIRQ().
     - In LPTIM IRQ handler, call HAL_LPTIM_IRQHandler().
2. Initialize the LPTIM HAL using HAL_LPTIM_Init(). This function configures mainly:
   - The instance: LPTIM1, LPTIM2, LPTIM3 or LPTIM4.
   - Clock: the counter clock.
     - Source : it can be either the ULPTIM input (IN1) or one of the internal clock; (APB, LSE, LSI or MSI).
     - Prescaler: select the clock divider.
   - UltraLowPowerClock : To be used only if the ULPTIM is selected as counter clock source.
     - Polarity: polarity of the active edge for the counter unit if the ULPTIM input is selected.
     - SampleTime: clock sampling time to configure the clock glitch filter.
   - Trigger: How the counter start.
     - Source: trigger can be software or one of the hardware triggers.
     - ActiveEdge : only for hardware trigger.
     - SampleTime : trigger sampling time to configure the trigger glitch filter.
   - OutputPolarity : 2 opposite polarities are possible.
   - UpdateMode: specifies whether the update of the autoreload and the compare values is done immediately or after the end of current period.
   - Input1Source: Source selected for input1 (GPIO or comparator output).
   - Input2Source: Source selected for input2 (GPIO or comparator output). Input2 is used only for encoder feature so is used only for LPTIM1 instance.
3. Six modes are available:
   - PWM Mode: To generate a PWM signal with specified period and pulse, call HAL_LPTIM_PWM_Start() or HAL_LPTIM_PWM_Start_IT() for interruption mode.
   - One Pulse Mode: To generate pulse with specified width in response to a stimulus, call HAL_LPTIM_OnePulse_Start() or HAL_LPTIM_OnePulse_Start_IT() for interruption mode.
   - Set once Mode: In this mode, the output changes the level (from low level to high level if the output polarity is configured high, else the opposite) when a compare match occurs. To start this mode, call HAL_LPTIM_SetOnce_Start() or HAL_LPTIM_SetOnce_Start_IT() for interruption mode.
   - Encoder Mode: To use the encoder interface call HAL_LPTIM_Encoder_Start() or HAL_LPTIM_Encoder_Start_IT() for interruption mode. Only available for LPTIM1 instance.
   - Time out Mode: an active edge on one selected trigger input rests the counter. The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart. To start this mode call HAL_LPTIM_TimeOut_Start_IT() or HAL_LPTIM_TimeOut_Start_IT() for interruption mode.
   - Counter Mode: counter can be used to count external events on the LPTIM Input1 or it can be used to count internal clock cycles. To start this mode, call HAL_LPTIM_Counter_Start() or HAL_LPTIM_Counter_Start_IT() for interruption mode.
4. User can stop any process by calling the corresponding API: HAL_LPTIM_Xxx_Stop() or HAL_LPTIM_Xxx_Stop_IT() if the process is already started in interruption mode.
5. De-initialize the LPTIM peripheral using HAL_LPTIM_DeInit().

Callback registration

The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_LPTIM_RegisterCallback() to register a callback. HAL_LPTIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_LPTIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_LPTIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- MspInitCallback : LPTIM Base Msp Init Callback.
- MspDeInitCallback : LPTIM Base Msp DeInit Callback.
- CompareMatchCallback : Compare match Callback.
- AutoReloadMatchCallback : Auto-reload match Callback.
- TriggerCallback : External trigger event detection Callback.
- CompareWriteCallback : Compare register write complete Callback.
- AutoReloadWriteCallback : Auto-reload register write complete Callback.
- DirectionUpCallback : Up-counting direction change Callback.
- DirectionDownCallback : Down-counting direction change Callback.
- UpdateEventCallback : Update event detection Callback.
- RepCounterWriteCallback : Repetition counter register write complete Callback.

By default, after the Init and when the state is HAL_LPTIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_LPTIM_TriggerCallback(), HAL_LPTIM_CompareMatchCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init/DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init/DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_LPTIM_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_LPTIM_STATE_READY or HAL_LPTIM_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LPTIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_LPTIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

45.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.
- DeInitialize the LPTIM peripheral.
- Initialize the LPTIM MSP.
- DeInitialize the LPTIM MSP.

This section contains the following APIs:

- ***HAL_LPTIM_Init()***
- ***HAL_LPTIM_DeInit()***
- ***HAL_LPTIM_MspInit()***
- ***HAL_LPTIM_MspDeInit()***

45.2.3 LPTIM Start Stop operation functions

This section provides functions allowing to:

- Start the PWM mode.
- Stop the PWM mode.
- Start the One pulse mode.
- Stop the One pulse mode.
- Start the Set once mode.
- Stop the Set once mode.
- Start the Encoder mode.
- Stop the Encoder mode.
- Start the Timeout mode.
- Stop the Timeout mode.
- Start the Counter mode.
- Stop the Counter mode.

This section contains the following APIs:

- *HAL_LPTIM_PWM_Start()*
- *HAL_LPTIM_PWM_Stop()*
- *HAL_LPTIM_PWM_Start_IT()*
- *HAL_LPTIM_PWM_Stop_IT()*
- *HAL_LPTIM_PWM_Start_DMA()*
- *HAL_LPTIM_PWM_Stop_DMA()*
- *HAL_LPTIM_OnePulse_Start()*
- *HAL_LPTIM_OnePulse_Stop()*
- *HAL_LPTIM_OnePulse_Start_IT()*
- *HAL_LPTIM_OnePulse_Stop_IT()*
- *HAL_LPTIM_SetOnce_Start()*
- *HAL_LPTIM_SetOnce_Stop()*
- *HAL_LPTIM_SetOnce_Start_IT()*
- *HAL_LPTIM_SetOnce_Stop_IT()*
- *HAL_LPTIM_Encoder_Start()*
- *HAL_LPTIM_Encoder_Stop()*
- *HAL_LPTIM_Encoder_Start_IT()*
- *HAL_LPTIM_Encoder_Stop_IT()*
- *HAL_LPTIM_TimeOut_Start()*
- *HAL_LPTIM_TimeOut_Stop()*
- *HAL_LPTIM_TimeOut_Start_IT()*
- *HAL_LPTIM_TimeOut_Stop_IT()*
- *HAL_LPTIM_Counter_Start()*
- *HAL_LPTIM_Counter_Stop()*
- *HAL_LPTIM_Counter_Start_IT()*
- *HAL_LPTIM_Counter_Stop_IT()*
- *HAL_LPTIM_IC_Start()*
- *HAL_LPTIM_IC_Stop()*
- *HAL_LPTIM_IC_Start_IT()*
- *HAL_LPTIM_IC_Stop_IT()*
- *HAL_LPTIM_IC_Start_DMA()*
- *HAL_LPTIM_IC_Stop_DMA()*
- *HAL_LPTIM_OC_ConfigChannel()*
- *HAL_LPTIM_IC_ConfigChannel()*

45.2.4 LPTIM Read operation functions

This section provides LPTIM Reading functions.

- Read the counter value.
- Read the period (Auto-reload) value.
- Read the pulse (Compare)value.

This section contains the following APIs:

- *HAL_LPTIM_ReadCounter()*
- *HAL_LPTIM_ReadAutoReload()*
- *HAL_LPTIM_ReadCapturedValue()*
- *HAL_LPTIM_IC_GetOffset()*

45.2.5 LPTIM Config function

This section provides LPTIM Config function.

- Configure channel: Output Compare mode, Period, Polarity.

This section contains the following APIs:

- *HAL_LPTIM_OC_ConfigChannel()*
- *HAL_LPTIM_IC_ConfigChannel()*

45.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_LPTIM_GetState()*

45.2.7 Detailed description of functions

HAL_LPTIM_Init

Function name

HAL_StatusTypeDef HAL_LPTIM_Init (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM according to the specified parameters in the LPTIM_InitTypeDef and initialize the associated handle.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_DeInit

Function name

HAL_StatusTypeDef HAL_LPTIM_DeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize the LPTIM peripheral.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_MspInit

Function name

void HAL_LPTIM_MspInit (LPTIM_HandleTypeDef * hlptim)

Function description

Initialize the LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_MspDeInit

Function name

void HAL_LPTIM_MspDeInit (LPTIM_HandleTypeDef * hlptim)

Function description

DeInitialize LPTIM MSP.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_LPTIM_OC_ConfigChannel (LPTIM_HandleTypeDef * hlptim, const LPTIM_OC_ConfigTypeDef * sConfig, uint32_t Channel)

Function description

Parameters

- **hlptim:** LPTIM handle
- **sConfig:** The output configuration structure
- **Channel:** LPTIM Channel to be configured This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

Notes

- Successive calls to HAL_LPTIM_OC_ConfigChannel can only be performed after a delay that must be greater or equal than the value of (PRESC x 3) kernel clock cycles, PRESC[2:0] being the clock decimal division factor (1, 2, 4, ..., 128). Any successive call violating this delay, leads to unpredictable results.

HAL_LPTIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM PWM generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM PWM generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Start_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel, const uint32_t * pData, uint32_t Length)

Function description

Start the LPTIM PWM generation in DMA mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected
- **pData:** The destination Buffer address
- **Length:** The length of data to be transferred from LPTIM peripheral to memory

Return values

- **HAL:** status

HAL_LPTIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_PWM_Stop_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM PWM generation in DMA mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM One pulse generation.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_OnePulse_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM One pulse generation in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM in Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM Set once mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Start the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_SetOnce_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_SetOnce_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stop the LPTIM Set once mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  – LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Start_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Encoder_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Encoder interface in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Timeout)

Function description

Start the Timeout function.

Parameters

- **hlptim:** LPTIM handle
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_TimeOut_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Timeout)

Function description

Start the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle
- **Timeout:** Specifies the TimeOut value to reset the counter. This parameter must be a value between 0x0000 and 0xFFFF.

Return values

- **HAL:** status

Notes

- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer restarts.

HAL_LPTIM_TimeOut_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_TimeOut_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Timeout function in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Start_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Start the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_Counter_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_Counter_Stop_IT (LPTIM_HandleTypeDef * hlptim)

Function description

Stop the Counter mode in interrupt mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** status

HAL_LPTIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_ConfigChannel (LPTIM_HandleTypeDef * hlptim, const LPTIM_IC_ConfigTypeDef * sConfig, uint32_t Channel)

Function description

Parameters

- **hlptim:** LPTIM handle
- **sConfig:** The input configuration structure
- **Channel:** LPTIM Channel to be configured This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **HAL:** status

Notes

- Successive calls to HAL_LPTIM_IC_ConfigChannel can only be performed after a delay that must be greater or equal than the value of (PRESC x 3) kernel clock cycles, PRESC[2:0] being the clock decimal division factor (1, 2, 4, ..., 128). Any successive call violating this delay, leads to unpredictable results.

HAL_LPTIM_IC_Start

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Starts the LPTIM Input Capture measurement.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stops the LPTIM Input Capture measurement.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: TIM Channel 1 selected
  – LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Starts the LPTIM Input Capture measurement in interrupt mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be enabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: TIM Channel 1 selected
  – LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop_IT (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stops the LPTIM Input Capture measurement in interrupt mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be disabled This parameter can be one of the following values:
  – LPTIM_CHANNEL_1: TIM Channel 1 selected
  – LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Start_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel, uint32_t * pData, uint32_t Length)

Function description

Starts the LPTIM Input Capture measurement in DMA mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be enabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected
- **pData:** The destination Buffer address
- **Length:** The length of data to be transferred from LPTIM peripheral to memory

Return values

- **HAL:** status

HAL_LPTIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_LPTIM_IC_Stop_DMA (LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Stops the LPTIM Input Capture measurement in DMA mode.

Parameters

- **hlptim:** LPTIM Input Capture handle
- **Channel:** LPTIM Channels to be disabled This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: TIM Channel 1 selected
  - LPTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

HAL_LPTIM_ReadCounter

Function name

uint32_t HAL_LPTIM_ReadCounter (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current counter value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Counter:** value.

HAL_LPTIM_ReadAutoReload

Function name

uint32_t HAL_LPTIM_ReadAutoReload (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the current Autoreload (Period) value.

Parameters

- **hlptim:** LPTIM handle

Return values

- **Autoreload:** value.

HAL_LPTIM_ReadCapturedValue

Function name

uint32_t HAL_LPTIM_ReadCapturedValue (const LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

Return the current Compare (Pulse) value.

Parameters

- **hlptim:** LPTIM handle
- **Channel:** LPTIM Channel to be selected This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **Compare:** value.

HAL_LPTIM_IC_GetOffset

Function name

uint8_t HAL_LPTIM_IC_GetOffset (const LPTIM_HandleTypeDef * hlptim, uint32_t Channel)

Function description

LPTimer Input Capture Get Offset(in counter step unit)

Parameters

- **hlptim:** pointer to a LPTIM_HandleTypeDef structure that contains the configuration information for LPTIM module.
- **Channel:** This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return values

- **The:** offset value

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset The Offset value is depending on the glitch filter value for the channel and the value of the prescaler for the kernel clock. Please check Errata Sheet V1_8 for more details under "variable latency on input capture channel" section.

HAL_LPTIM_IRQHandler

Function name

void HAL_LPTIM_IRQHandler (LPTIM_HandleTypeDef * hlptim)

Function description

Handle LPTIM interrupt request.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareMatchCallback

Function name

void HAL_LPTIM_CompareMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadMatchCallback

Function name

void HAL_LPTIM_AutoReloadMatchCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload match callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_TriggerCallback

Function name

void HAL_LPTIM_TriggerCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Trigger detected callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_CompareWriteCallback

Function name

void HAL_LPTIM_CompareWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Compare write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_AutoReloadWriteCallback

Function name

void HAL_LPTIM_AutoReloadWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Autoreload write callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionUpCallback

Function name

void HAL_LPTIM_DirectionUpCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Down to Up callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_DirectionDownCallback

Function name

void HAL_LPTIM_DirectionDownCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Direction counter changed from Up to Down callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_ErrorCallback

Function name

void HAL_LPTIM_ErrorCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Error callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_UpdateEventCallback

Function name

void HAL_LPTIM_UpdateEventCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Repetition counter underflowed (or contains zero) and LPTIM counter overflowed callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_UpdateEventHalfCpltCallback

Function name

void HAL_LPTIM_UpdateEventHalfCpltCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Update event half complete callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_RepCounterWriteCallback

Function name

void HAL_LPTIM_RepCounterWriteCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Successful APB bus write to repetition counter register callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_IC_CaptureCallback

Function name

void HAL_LPTIM_IC_CaptureCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Input Capture callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_IC_CaptureHalfCpltCallback

Function name

void HAL_LPTIM_IC_CaptureHalfCpltCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Input Capture half complete callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM IC handle

Return values

- **None:**

HAL_LPTIM_IC_OverCaptureCallback

Function name

void HAL_LPTIM_IC_OverCaptureCallback (LPTIM_HandleTypeDef * hlptim)

Function description

Over Capture callback in non-blocking mode.

Parameters

- **hlptim:** LPTIM handle

Return values

- **None:**

HAL_LPTIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_RegisterCallback (LPTIM_HandleTypeDef * hlptim, HAL_LPTIM_CallbackIDTypeDef CallbackID, pLPTIM_CallbackTypeDef pCallback)

Function description

Register a User LPTIM callback to be used instead of the weak predefined callback.

Parameters

- **hlptim:** LPTIM handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_LPTIM_MSPINIT_CB_ID LPTIM Base Msp Init Callback ID
  - HAL_LPTIM_MSPDEINIT_CB_ID LPTIM Base Msp DeInit Callback ID
  - HAL_LPTIM_COMPARE_MATCH_CB_ID Compare match Callback ID
  - HAL_LPTIM_AUTORELOAD_MATCH_CB_ID Auto-reload match Callback ID
  - HAL_LPTIM_TRIGGER_CB_ID External trigger event detection Callback ID
  - HAL_LPTIM_COMPARE_WRITE_CB_ID Compare register write complete Callback ID
  - HAL_LPTIM_AUTORELOAD_WRITE_CB_ID Auto-reload register write complete Callback ID
  - HAL_LPTIM_DIRECTION_UP_CB_ID Up-counting direction change Callback ID
  - HAL_LPTIM_DIRECTION_DOWN_CB_ID Down-counting direction change Callback ID
  - HAL_LPTIM_UPDATE_EVENT_CB_ID Update event detection Callback ID
  - HAL_LPTIM_REP_COUNTER_WRITE_CB_ID Repetition counter register write complete Callback ID
  - HAL_LPTIM_UPDATE_EVENT_HALF_CB_ID Update event Half detection Callback ID
  - HAL_LPTIM_ERROR_CB_ID Error Callback ID
  - HAL_LPTIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_LPTIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_LPTIM_OVER_CAPTURE_CB_ID Over Capture Callback ID
- **pCallback:** pointer to the callback function

Return values

- **status:**

HAL_LPTIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LPTIM_UnRegisterCallback (LPTIM_HandleTypeDef * hlptim, HAL_LPTIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a LPTIM callback LLPTIM callback is redirected to the weak predefined callback.

Parameters

- **hlptim:** LPTIM handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_LPTIM_MSPINIT_CB_ID LPTIM Base Msp Init Callback ID
  - HAL_LPTIM_MSPDEINIT_CB_ID LPTIM Base Msp DeInit Callback ID
  - HAL_LPTIM_COMPARE_MATCH_CB_ID Compare match Callback ID
  - HAL_LPTIM_AUTORELOAD_MATCH_CB_ID Auto-reload match Callback ID
  - HAL_LPTIM_TRIGGER_CB_ID External trigger event detection Callback ID
  - HAL_LPTIM_COMPARE_WRITE_CB_ID Compare register write complete Callback ID
  - HAL_LPTIM_AUTORELOAD_WRITE_CB_ID Auto-reload register write complete Callback ID
  - HAL_LPTIM_DIRECTION_UP_CB_ID Up-counting direction change Callback ID
  - HAL_LPTIM_DIRECTION_DOWN_CB_ID Down-counting direction change Callback ID
  - HAL_LPTIM_UPDATE_EVENT_CB_ID Update event detection Callback ID
  - HAL_LPTIM_REP_COUNTER_WRITE_CB_ID Repetition counter register write complete Callback ID
  - HAL_LPTIM_UPDATE_EVENT_HALF_CB_ID Update event Half detection Callback ID
  - HAL_LPTIM_ERROR_CB_ID Error Callback ID
  - HAL_LPTIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_LPTIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_LPTIM_OVER_CAPTURE_CB_ID Over Capture Callback ID

Return values

- **status:**

HAL_LPTIM_GetState

Function name

HAL_LPTIM_StateTypeDef HAL_LPTIM_GetState (const LPTIM_HandleTypeDef * hlptim)

Function description

Return the LPTIM handle state.

Parameters

- **hlptim:** LPTIM handle

Return values

- **HAL:** state

45.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

45.3.1 LPTIM

LPTIM

LPTIM Channel

LPTIM_CHANNEL_1

Capture/compare channel 1 identifier

LPTIM_CHANNEL_2

Capture/compare channel 2 identifier

LPTIM Clock Polarity

LPTIM_CLOCKPOLARITY_RISING

LPTIM_CLOCKPOLARITY_FALLING

LPTIM_CLOCKPOLARITY_RISING_FALLING

LPTIM Clock Prescaler

LPTIM_PRESCALER_DIV1

LPTIM_PRESCALER_DIV2

LPTIM_PRESCALER_DIV4

LPTIM_PRESCALER_DIV8

LPTIM_PRESCALER_DIV16

LPTIM_PRESCALER_DIV32

LPTIM_PRESCALER_DIV64

LPTIM_PRESCALER_DIV128

LPTIM Clock Sample Time

LPTIM_CLOCKSAMPLETIME_DIRECTTRANSITION

LPTIM_CLOCKSAMPLETIME_2TRANSITIONS

LPTIM_CLOCKSAMPLETIME_4TRANSITIONS

LPTIM_CLOCKSAMPLETIME_8TRANSITIONS

LPTIM Clock Source

LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC

LPTIM_CLOCKSOURCE_ULPTIM

LPTIM Counter Source

LPTIM_COUNTERSOURCE_INTERNAL

LPTIM_COUNTERSOURCE_EXTERNAL

LPTIM DMA Handle Index

LPTIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

LPTIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Update event 1 DMA request

LPTIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Update event 2 DMA request

LPTIM DMA Sources

LPTIM_DMA_UPDATE

DMA request is triggered by the update event

LPTIM_DMA_CC1

DMA request is triggered by the capture 1 event

LPTIM_DMA_CC2

DMA request is triggered by the capture 2 event

LPTIM Exported Macros

__HAL_LPTIM_RESET_HANDLE_STATE

Description:
- Reset LPTIM handle state.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_ENABLE

Description:
- Enable the LPTIM peripheral.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_DISABLE

Description:
- Disable the LPTIM peripheral.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_START_CONTINUOUS

Description:
- Start the LPTIM peripheral in Continuous mode.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_START_SINGLE

Description:
- Start the LPTIM peripheral in single mode.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_RESET_COUNTER

Description:
- Reset the LPTIM Counter register in synchronous mode.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_RESET_COUNTER_AFTERREAD

Description:
- Reset after read of the LPTIM Counter register in asynchronous mode.

Parameters:
- __HANDLE__: LPTIM handle

Return value:
- None

__HAL_LPTIM_AUTORELOAD_SET

Description:
- Write the passed parameter in the Autoreload register.

Parameters:
- __HANDLE__: LPTIM handle
- __VALUE__: Autoreload value

Return value:
- None

Notes:
- The ARR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_COMPARE_SET

Description:
- Write the passed parameter in the Compare register.

Parameters:
- __HANDLE__: LPTIM handle
- __VALUE__: Compare value
- __CHANNEL__: TIM Channel to be configured

Return value:
- None

Notes:
- The CCRx registers can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_REPETITIONCOUNTER_SET

Description:
- Write the passed parameter in the Repetition register.

Parameters:
- __HANDLE__: LPTIM handle
- __VALUE__: Repetition value

Return value:
- None

__HAL_LPTIM_REPETITIONCOUNTER_GET

Description:

- Return the current Repetition value.

Parameters:

- __HANDLE__: LPTIM handle

Return value:

- Repetition: register value

Notes:

- The RCR register can only be modified when the LPTIM instance is enabled.

__HAL_LPTIM_CAPTURE_COMPARE_ENABLE

Description:

- Enable the LPTIM signal input/output on the corresponding pin.

Parameters:

- __HANDLE__: LPTIM handle
- __CHANNEL__: LPTIM Channels to be enabled. This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return value:

- None

__HAL_LPTIM_CAPTURE_COMPARE_DISABLE

Description:

- Disable the LPTIM signal input/output on the corresponding pin.

Parameters:

- __HANDLE__: LPTIM handle
- __CHANNEL__: LPTIM Channels to be disabled. This parameter can be one of the following values:
  - LPTIM_CHANNEL_1: LPTIM Channel 1 selected
  - LPTIM_CHANNEL_2: LPTIM Channel 2 selected

Return value:

- None

__HAL_LPTIM_GET_FLAG

Description:

- Check whether the specified LPTIM flag is set or not.

Parameters:

- __HANDLE__: LPTIM handle
- __FLAG__: LPTIM flag to check This parameter can be a value of:
  - LPTIM_FLAG_REPOK : Repetition register update OK Flag.
  - LPTIM_FLAG_UPDATE : Update event Flag.
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMP1OK : Compare register 1 update OK Flag.
  - LPTIM_FLAG_CMP2OK : Compare register 2 update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CC1 : Capture/Compare 1 interrupt flag.
  - LPTIM_FLAG_CC2 : Capture/Compare 2 interrupt flag.
  - LPTIM_FLAG_CC1O : Capture/Compare 1 over-capture flag.
  - LPTIM_FLAG_CC2O : Capture/Compare 2 over-capture flag.
  - LPTIM_FLAG_DIEROK : DMA & interrupt enable update OK flag.

Return value:

- The: state of the specified flag (SET or RESET).

__HAL_LPTIM_CLEAR_FLAG

Description:

- Clear the specified LPTIM flag.

Parameters:

- __HANDLE__: LPTIM handle.
- __FLAG__: LPTIM flag to clear. This parameter can be a value of:
  - LPTIM_FLAG_REPOK : Repetition register update OK Flag.
  - LPTIM_FLAG_UPDATE : Update event Flag.
  - LPTIM_FLAG_DOWN : Counter direction change up Flag.
  - LPTIM_FLAG_UP : Counter direction change down to up Flag.
  - LPTIM_FLAG_ARROK : Autoreload register update OK Flag.
  - LPTIM_FLAG_CMP1OK : Compare register 1 update OK Flag.
  - LPTIM_FLAG_CMP2OK : Compare register 2 update OK Flag.
  - LPTIM_FLAG_EXTTRIG : External trigger edge event Flag.
  - LPTIM_FLAG_ARRM : Autoreload match Flag.
  - LPTIM_FLAG_CC1 : Capture/Compare 1 interrupt flag.
  - LPTIM_FLAG_CC2 : Capture/Compare 2 interrupt flag.
  - LPTIM_FLAG_CC1O : Capture/Compare 1 over-capture flag.
  - LPTIM_FLAG_CC2O : Capture/Compare 2 over-capture flag.
  - LPTIM_FLAG_DIEROK : DMA & interrupt enable update OK flag.

Return value:

- None.

__HAL_LPTIM_ENABLE_IT

Description:
- Enable the specified LPTIM interrupt.

Parameters:
- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:
- None.

Notes:
- The LPTIM interrupts can only be enabled when the LPTIM instance is enabled.

__HAL_LPTIM_DISABLE_IT

Description:
- Disable the specified LPTIM interrupt.

Parameters:
- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to set. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:
- None.

Notes:
- The LPTIM interrupts can only be disabled when the LPTIM instance is enabled.

__HAL_LPTIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the LPTIM DMA request to enable. This parameter can be one of the following values:
  - LPTIM_DMA_UPDATE: Update DMA request
  - LPTIM_DMA_CC1: Capture/Compare 1 DMA request
  - LPTIM_DMA_CC2: Capture/Compare 2 DMA request

Return value:

- None

__HAL_LPTIM_DISABLE_DMA

Description:

- Disable the specified DMA request.

Parameters:

- __HANDLE__: specifies the LPTIM Handle.
- __DMA__: specifies the LPTIM DMA request to disable. This parameter can be one of the following values:
  - LPTIM_DMA_UPDATE: Update DMA request
  - LPTIM_DMA_CC1: Capture/Compare 1 DMA request
  - LPTIM_DMA_CC2: Capture/Compare 2 DMA request

Return value:

- None

__HAL_LPTIM_GET_IT_SOURCE

Description:

- Check whether the specified LPTIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: LPTIM handle.
- __INTERRUPT__: LPTIM interrupt to check. This parameter can be a value of:
  - LPTIM_IT_REPOK : Repetition register update OK Interrupt.
  - LPTIM_IT_UPDATE : Update event register Interrupt.
  - LPTIM_IT_DOWN : Counter direction change up Interrupt.
  - LPTIM_IT_UP : Counter direction change down to up Interrupt.
  - LPTIM_IT_ARROK : Autoreload register update OK Interrupt.
  - LPTIM_IT_CMP1OK : Compare register 1 update OK Interrupt.
  - LPTIM_IT_CMP2OK : Compare register 2 update OK Interrupt.
  - LPTIM_IT_EXTTRIG : External trigger edge event Interrupt.
  - LPTIM_IT_ARRM : Autoreload match Interrupt.
  - LPTIM_IT_CC1 : Capture/Compare 1 interrupt Interrupt.
  - LPTIM_IT_CC2 : Capture/Compare 2 interrupt Interrupt.
  - LPTIM_IT_CC1O : Capture/Compare 1 over-capture Interrupt.
  - LPTIM_IT_CC2O : Capture/Compare 2 over-capture Interrupt.

Return value:

- Interrupt: status.

LPTIM External Trigger Polarity

LPTIM_ACTIVEEDGE_RISING

LPTIM_ACTIVEEDGE_FALLING

LPTIM_ACTIVEEDGE_RISING_FALLING

LPTIM Flags Definition

LPTIM_FLAG_CC1O

LPTIM_FLAG_CC2O

LPTIM_FLAG_CC1

LPTIM_FLAG_CC2

LPTIM_FLAG_CMP1OK

LPTIM_FLAG_CMP2OK

LPTIM_FLAG_DIEROK

LPTIM_FLAG_REPOK

LPTIM_FLAG_UPDATE

LPTIM_FLAG_DOWN

LPTIM_FLAG_UP

LPTIM_FLAG_ARROK

LPTIM_FLAG_EXTTRIG

LPTIM_FLAG_ARRM

LPTIM Input1 Source

LPTIM_INPUT1SOURCE_GPIO

For LPTIM1, LPTIM2, LPTIM3 and LPTIM4

LPTIM_INPUT1SOURCE_COMP1

For LPTIM1, LPTIM2, LPTIM3 and LPTIM4

LPTIM Input2 Source

LPTIM_INPUT2SOURCE_GPIO

For LPTIM1 and LPTIM2

LPTIM_INPUT2SOURCE_COMP2

For LPTIM1 and LPTIM2

LPTIM Input Capture Filter

LPTIM_ICFLT_CLOCK_DIV1

any external input capture signal level change is considered as a valid transition

LPTIM_ICFLT_CLOCK_DIV2

external input capture signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LPTIM_ICFLT_CLOCK_DIV4

external input capture signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LPTIM_ICFLT_CLOCK_DIV8

external input capture signal level change must be stable for at least 8 clock periods before it is considered as valid transition

LPTIM Input Capture Polarity

LPTIM_ICPOLARITY_RISING

Capture/Compare input rising polarity

LPTIM_ICPOLARITY_FALLING

Capture/Compare input falling polarity

LPTIM_ICPOLARITY_RISING_FALLING

Capture/Compare input rising and falling polarities

LPTIM Input Capture Prescaler

LPTIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

LPTIM_ICPSC_DIV2

Capture performed once every 2 events

LPTIM_ICPSC_DIV4

Capture performed once every 4 events

LPTIM_ICPSC_DIV8

Capture performed once every 8 events

LPTIM Input Capture Source

LPTIM_IC1SOURCE_GPIO

For LPTIM1, LPTIM2 and LPTIM3

LPTIM_IC1SOURCE_COMP1

For LPTIM1, LPTIM2 and LPTIM3

LPTIM_IC1SOURCE_COMP2

For LPTIM1, LPTIM2 and LPTIM3

LPTIM_IC2SOURCE_GPIO

For LPTIM1, LPTIM2 and LPTIM3

LPTIM_IC2SOURCE_LSI

For LPTIM1

LPTIM_IC2SOURCE_LSE

For LPTIM1

LPTIM_IC2SOURCE_HSI_256

For LPTIM2

LPTIM_IC2SOURCE_MSIS_1024

For LPTIM2

LPTIM_IC2SOURCE_MSIS_4

For LPTIM2

LPTIM Interrupts Definition

LPTIM_IT_CC1O

LPTIM_IT_CC2O

LPTIM_IT_CC1

LPTIM_IT_CC2

LPTIM_IT_CMP1OK

LPTIM_IT_CMP2OK

LPTIM_IT_REPOK

LPTIM_IT_UPDATE

LPTIM_IT_DOWN

LPTIM_IT_UP

LPTIM_IT_ARROK

LPTIM_IT_EXTTRIG

LPTIM_IT_ARRM

LPTIM Output Compare Polarity

LPTIM_OCPOLARITY_HIGH

Capture/Compare output polarity

LPTIM_OCPOLARITY_LOW

Capture/Compare output polarity

LPTIM Trigger Sample Time

LPTIM_TRIGSAMPLETIME_DIRECTTRANSITION

LPTIM_TRIGSAMPLETIME_2TRANSITIONS

LPTIM_TRIGSAMPLETIME_4TRANSITIONS

LPTIM_TRIGSAMPLETIME_8TRANSITIONS

LPTIM Trigger Source

LPTIM_TRIGSOURCE_SOFTWARE

LPTIM_TRIGSOURCE_0

LPTIM_TRIGSOURCE_1

LPTIM_TRIGSOURCE_2

LPTIM_TRIGSOURCE_3

LPTIM_TRIGSOURCE_4

LPTIM_TRIGSOURCE_5

LPTIM_TRIGSOURCE_6

LPTIM_TRIGSOURCE_7

LPTIM Updating Mode

LPTIM_UPDATE_IMMEDIATE

LPTIM_UPDATE_ENDOFPERIOD

46 HAL LTDC Generic Driver

46.1 LTDC Firmware driver registers structures

46.1.1 LTDC_ColorTypeDef

LTDC_ColorTypeDef is defined in the stm32u5xx_hal_ltdc.h

Data Fields

- ***uint8_t Blue***
- ***uint8_t Green***
- ***uint8_t Red***
- ***uint8_t Reserved***

Field Documentation

- ***uint8_t LTDC_ColorTypeDef::Blue***
  Configures the blue value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Green***
  Configures the green value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Red***
  Configures the red value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint8_t LTDC_ColorTypeDef::Reserved***
  Reserved 0xFF

46.1.2 LTDC_InitTypeDef

LTDC_InitTypeDef is defined in the stm32u5xx_hal_ltdc.h

Data Fields

- ***uint32_t HSPolarity***
- ***uint32_t VSPolarity***
- ***uint32_t DEPolarity***
- ***uint32_t PCPolarity***
- ***uint32_t HorizontalSync***
- ***uint32_t VerticalSync***
- ***uint32_t AccumulatedHBP***
- ***uint32_t AccumulatedVBP***
- ***uint32_t AccumulatedActiveW***
- ***uint32_t AccumulatedActiveH***
- ***uint32_t TotalWidth***
- ***uint32_t TotalHeigh***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_InitTypeDef::HSPolarity***
  configures the horizontal synchronization polarity. This parameter can be one value of *LTDC_HS_POLARITY*
- ***uint32_t LTDC_InitTypeDef::VSPolarity***
  configures the vertical synchronization polarity. This parameter can be one value of *LTDC_VS_POLARITY*
- ***uint32_t LTDC_InitTypeDef::DEPolarity***
  configures the data enable polarity. This parameter can be one of value of *LTDC_DE_POLARITY*
- ***uint32_t LTDC_InitTypeDef::PCPolarity***
  configures the pixel clock polarity. This parameter can be one of value of *LTDC_PC_POLARITY*

- ***uint32_t LTDC_InitTypeDef::HorizontalSync***
  configures the number of Horizontal synchronization width. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::VerticalSync***
  configures the number of Vertical synchronization height. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedHBP***
  configures the accumulated horizontal back porch width. This parameter must be a number between Min_Data = LTDC_HorizontalSync and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedVBP***
  configures the accumulated vertical back porch height. This parameter must be a number between Min_Data = LTDC_VerticalSync and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveW***
  configures the accumulated active width. This parameter must be a number between Min_Data = LTDC_AccumulatedHBP and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::AccumulatedActiveH***
  configures the accumulated active height. This parameter must be a number between Min_Data = LTDC_AccumulatedVBP and Max_Data = 0x7FF.
- ***uint32_t LTDC_InitTypeDef::TotalWidth***
  configures the total width. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveW and Max_Data = 0xFFF.
- ***uint32_t LTDC_InitTypeDef::TotalHeigh***
  configures the total height. This parameter must be a number between Min_Data = LTDC_AccumulatedActiveH and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_InitTypeDef::Backcolor***
  Configures the background color.

46.1.3 LTDC_LayerCfgTypeDef

LTDC_LayerCfgTypeDef is defined in the stm32u5xx_hal_ltdc.h

Data Fields

- ***uint32_t WindowX0***
- ***uint32_t WindowX1***
- ***uint32_t WindowY0***
- ***uint32_t WindowY1***
- ***uint32_t PixelFormat***
- ***uint32_t Alpha***
- ***uint32_t Alpha0***
- ***uint32_t BlendingFactor1***
- ***uint32_t BlendingFactor2***
- ***uint32_t FBStartAdress***
- ***uint32_t ImageWidth***
- ***uint32_t ImageHeight***
- ***LTDC_ColorTypeDef Backcolor***

Field Documentation

- ***uint32_t LTDC_LayerCfgTypeDef::WindowX0***
  Configures the Window Horizontal Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowX1***
  Configures the Window Horizontal Stop Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF.
- ***uint32_t LTDC_LayerCfgTypeDef::WindowY0***
  Configures the Window vertical Start Position. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.

- ***uint32_t LTDC_LayerCfgTypeDef::WindowY1***
  Configures the Window vertical Stop Position. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x7FF.
- ***uint32_t LTDC_LayerCfgTypeDef::PixelFormat***
  Specifies the pixel format. This parameter can be one of value of ***LTDC_Pixelformat***
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha***
  Specifies the constant alpha used for blending. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::Alpha0***
  Configures the default alpha value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor1***
  Select the blending factor 1. This parameter can be one of value of ***LTDC_BlendingFactor1***
- ***uint32_t LTDC_LayerCfgTypeDef::BlendingFactor2***
  Select the blending factor 2. This parameter can be one of value of ***LTDC_BlendingFactor2***
- ***uint32_t LTDC_LayerCfgTypeDef::FBStartAdress***
  Configures the color frame buffer address
- ***uint32_t LTDC_LayerCfgTypeDef::ImageWidth***
  Configures the color frame buffer line length. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x1FFF.
- ***uint32_t LTDC_LayerCfgTypeDef::ImageHeight***
  Specifies the number of line in frame buffer. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0x7FF.
- ***LTDC_ColorTypeDef LTDC_LayerCfgTypeDef::Backcolor***
  Configures the layer background color.

46.1.4 __LTDC_HandleTypeDef

__LTDC_HandleTypeDef is defined in the stm32u5xx_hal_ltdc.h

Data Fields

- ***LTDC_TypeDef * Instance***
- ***LTDC_InitTypeDef Init***
- ***LTDC_LayerCfgTypeDef LayerCfg***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LTDC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* LineEventCallback***
- ***void(* ReloadEventCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***LTDC_TypeDef* __LTDC_HandleTypeDef::Instance***
  LTDC Register base address
- ***LTDC_InitTypeDef __LTDC_HandleTypeDef::Init***
  LTDC parameters
- ***LTDC_LayerCfgTypeDef __LTDC_HandleTypeDef::LayerCfg[MAX_LAYER]***
  LTDC Layers parameters
- ***HAL_LockTypeDef __LTDC_HandleTypeDef::Lock***
  LTDC Lock
- ***__IO HAL_LTDC_StateTypeDef __LTDC_HandleTypeDef::State***
  LTDC state
- ***__IO uint32_t __LTDC_HandleTypeDef::ErrorCode***
  LTDC Error code

- ***void(* __LTDC_HandleTypeDef::LineEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Line Event Callback
- ***void(* __LTDC_HandleTypeDef::ReloadEventCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Reload Event Callback
- ***void(* __LTDC_HandleTypeDef::ErrorCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Error Callback
- ***void(* __LTDC_HandleTypeDef::MspInitCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Msp Init callback
- ***void(* __LTDC_HandleTypeDef::MspDeInitCallback)(struct __LTDC_HandleTypeDef *hltdc)***
  LTDC Msp DeInit callback

46.2 LTDC Firmware driver API description

The following section lists the various functions of the LTDC library.

46.2.1 How to use this driver

The LTDC HAL driver can be used as follows:

1. Declare a LTDC_HandleTypeDef handle structure, for example: LTDC_HandleTypeDef hltdc;
2. Initialize the LTDC low level resources by implementing the HAL_LTDC_MspInit() API:
   a. Enable the LTDC interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the LTDC interrupt priority
      - Enable the NVIC LTDC IRQ Channel
3. Initialize the required configuration through the following parameters: the LTDC timing, the horizontal and vertical polarity, the pixel clock polarity, Data Enable polarity and the LTDC background color value using HAL_LTDC_Init() function

Configuration

1. Program the required configuration through the following parameters: the pixel format, the blending factors, input alpha value, the window size and the image size using HAL_LTDC_ConfigLayer() function for foreground or/and background layer.
2. Optionally, configure and enable the CLUT using HAL_LTDC_ConfigCLUT() and HAL_LTDC_EnableCLUT functions.
3. Optionally, enable the Dither using HAL_LTDC_EnableDither().
4. Optionally, configure and enable the Color keying using HAL_LTDC_ConfigColorKeying() and HAL_LTDC_EnableColorKeying functions.
5. Optionally, configure LineInterrupt using HAL_LTDC_ProgramLineEvent() function
6. If needed, reconfigure and change the pixel format value, the alpha value value, the window size, the window position and the layer start address for foreground or/and background layer using respectively the following functions: HAL_LTDC_SetPixelFormat(), HAL_LTDC_SetAlpha(), HAL_LTDC_SetWindowSize(), HAL_LTDC_SetWindowPosition() and HAL_LTDC_SetAddress().
7. Variant functions with _NoReload suffix allows to set the LTDC configuration/settings without immediate reload. This is useful in case when the program requires to modify serval LTDC settings (on one or both layers) then applying(reload) these settings in one shot by calling the function HAL_LTDC_Reload(). After calling the _NoReload functions to set different color/format/layer settings, the program shall call the function HAL_LTDC_Reload() to apply(reload) these settings. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_IMMEDIATE if an immediate reload is required. Function HAL_LTDC_Reload() can be called with the parameter ReloadType set to LTDC_RELOAD_VERTICAL_BLANKING if the reload should be done in the next vertical blanking period, this option allows to avoid display flicker by applying the new settings during the vertical blanking period.
8. To control LTDC state you can use the following function: HAL_LTDC_GetState()

LTDC HAL driver macros list

Below the list of most used macros in LTDC HAL driver.

- __HAL_LTDC_ENABLE: Enable the LTDC.

- __HAL_LTDC_DISABLE: Disable the LTDC.
- __HAL_LTDC_LAYER_ENABLE: Enable an LTDC Layer.
- __HAL_LTDC_LAYER_DISABLE: Disable an LTDC Layer.
- __HAL_LTDC_RELOAD_IMMEDIATE_CONFIG: Reload Layer Configuration.
- __HAL_LTDC_GET_FLAG: Get the LTDC pending flags.
- __HAL_LTDC_CLEAR_FLAG: Clear the LTDC pending flags.
- __HAL_LTDC_ENABLE_IT: Enable the specified LTDC interrupts.
- __HAL_LTDC_DISABLE_IT: Disable the specified LTDC interrupts.
- __HAL_LTDC_GET_IT_SOURCE: Check whether the specified LTDC interrupt has occurred or not.

Note: You can refer to the LTDC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_LTDC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use function HAL_LTDC_RegisterCallback() to register a callback.

Function HAL_LTDC_RegisterCallback() allows to register following callbacks:

- LineEventCallback : LTDC Line Event Callback.
- ReloadEventCallback : LTDC Reload Event Callback.
- ErrorCallback : LTDC Error Callback
- MspInitCallback : LTDC MspInit.
- MspDeInitCallback : LTDC MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_LTDC_UnRegisterCallback() to reset a callback to the default weak function. HAL_LTDC_UnRegisterCallback() takes as parameters the HAL peripheral handle and the callback ID.

This function allows to reset following callbacks:

- LineEventCallback : LTDC Line Event Callback
- ReloadEventCallback : LTDC Reload Event Callback
- ErrorCallback : LTDC Error Callback
- MspInitCallback : LTDC MspInit
- MspDeInitCallback : LTDC MspDeInit.

By default, after the HAL_LTDC_Init and when the state is HAL_LTDC_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_LTDC_LineEventCallback(), HAL_LTDC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak (surcharged) functions in the HAL_LTDC_Init() and HAL_LTDC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_LTDC_Init() and HAL_LTDC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_LTDC_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_LTDC_STATE_READY or HAL_LTDC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_LTDC_RegisterCallback() before calling HAL_LTDC_DeInit() or HAL_LTDC_Init() function.

When the compilation define USE_HAL_LTDC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

46.2.2 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC
- De-initialize the LTDC

This section contains the following APIs:

- ***HAL_LTDC_Init()***
- ***HAL_LTDC_DeInit()***
- ***HAL_LTDC_MspInit()***

- *HAL_LTDC_MspDeInit()*
- *HAL_LTDC_RegisterCallback()*
- *HAL_LTDC_UnRegisterCallback()*
- *HAL_LTDC_ErrorCallback()*
- *HAL_LTDC_LineEventCallback()*
- *HAL_LTDC_ReloadEventCallback()*

46.2.3 IO operation functions

This section provides function allowing to:

- Handle LTDC interrupt request

This section contains the following APIs:

- *HAL_LTDC_IRQHandler()*
- *HAL_LTDC_ErrorCallback()*
- *HAL_LTDC_LineEventCallback()*
- *HAL_LTDC_ReloadEventCallback()*

46.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure the LTDC foreground or/and background parameters.
- Set the active layer.
- Configure the color keying.
- Configure the C-LUT.
- Enable / Disable the color keying.
- Enable / Disable the C-LUT.
- Update the layer position.
- Update the layer size.
- Update pixel format on the fly.
- Update transparency on the fly.
- Update address on the fly.

This section contains the following APIs:

- *HAL_LTDC_ConfigLayer()*
- *HAL_LTDC_ConfigColorKeying()*
- *HAL_LTDC_ConfigCLUT()*
- *HAL_LTDC_EnableColorKeying()*
- *HAL_LTDC_DisableColorKeying()*
- *HAL_LTDC_EnableCLUT()*
- *HAL_LTDC_DisableCLUT()*
- *HAL_LTDC_EnableDither()*
- *HAL_LTDC_DisableDither()*
- *HAL_LTDC_SetWindowSize()*
- *HAL_LTDC_SetWindowPosition()*
- *HAL_LTDC_SetPixelFormat()*
- *HAL_LTDC_SetAlpha()*
- *HAL_LTDC_SetAddress()*
- *HAL_LTDC_SetPitch()*
- *HAL_LTDC_ProgramLineEvent()*
- *HAL_LTDC_Reload()*
- *HAL_LTDC_ConfigLayer_NoReload()*
- *HAL_LTDC_SetWindowSize_NoReload()*

- *HAL_LTDC_SetWindowPosition_NoReload()*
- *HAL_LTDC_SetPixelFormat_NoReload()*
- *HAL_LTDC_SetAlpha_NoReload()*
- *HAL_LTDC_SetAddress_NoReload()*
- *HAL_LTDC_SetPitch_NoReload()*
- *HAL_LTDC_ConfigColorKeying_NoReload()*
- *HAL_LTDC_EnableColorKeying_NoReload()*
- *HAL_LTDC_DisableColorKeying_NoReload()*
- *HAL_LTDC_EnableCLUT_NoReload()*
- *HAL_LTDC_DisableCLUT_NoReload()*

46.2.5 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the LTDC handle state.
- Get the LTDC handle error code.

This section contains the following APIs:

- *HAL_LTDC_GetState()*
- *HAL_LTDC_GetError()*

46.2.6 Detailed description of functions

HAL_LTDC_Init

Function name

HAL_StatusTypeDef HAL_LTDC_Init (LTDC_HandleTypeDef * hltdc)

Function description

Initialize the LTDC according to the specified parameters in the LTDC_InitTypeDef.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_DeInit

Function name

HAL_StatusTypeDef HAL_LTDC_DeInit (LTDC_HandleTypeDef * hltdc)

Function description

De-initialize the LTDC peripheral.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_MspInit

Function name

void HAL_LTDC_MspInit (LTDC_HandleTypeDef * hltdc)

Function description

Initialize the LTDC MSP.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_MspDeInit

Function name

void HAL_LTDC_MspDeInit (LTDC_HandleTypeDef * hltdc)

Function description

De-initialize the LTDC MSP.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_ErrorCallback

Function name

void HAL_LTDC_ErrorCallback (LTDC_HandleTypeDef * hltdc)

Function description

Error LTDC callback.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_LineEventCallback

Function name

void HAL_LTDC_LineEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

Line Event callback.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_ReloadEventCallback

Function name

void HAL_LTDC_ReloadEventCallback (LTDC_HandleTypeDef * hltdc)

Function description

Reload Event callback.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **None:**

HAL_LTDC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_LTDC_RegisterCallback (LTDC_HandleTypeDef * hltdc, HAL_LTDC_CallbackIDTypeDef CallbackID, pLTDC_CallbackTypeDef pCallback)

Function description

Register a User LTDC Callback To be used instead of the weak predefined callback.

Parameters

- **hltdc:** ltdc handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_LTDC_LINE_EVENT_CB_ID Line Event Callback ID
  - HAL_LTDC_RELOAD_EVENT_CB_ID Reload Event Callback ID
  - HAL_LTDC_ERROR_CB_ID Error Callback ID
  - HAL_LTDC_MSPINIT_CB_ID MspInit callback ID
  - HAL_LTDC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_LTDC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_LTDC_UnRegisterCallback (LTDC_HandleTypeDef * hltdc, HAL_LTDC_CallbackIDTypeDef CallbackID)

Function description

Unregister an LTDC Callback LTDC callback is redirected to the weak predefined callback.

Parameters

- **hltdc:** ltdc handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_LTDC_LINE_EVENT_CB_ID Line Event Callback ID
  - HAL_LTDC_RELOAD_EVENT_CB_ID Reload Event Callback ID
  - HAL_LTDC_ERROR_CB_ID Error Callback ID
  - HAL_LTDC_MSPINIT_CB_ID MspInit callback ID
  - HAL_LTDC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **status:**

HAL_LTDC_IRQHandler

Function name

void HAL_LTDC_IRQHandler (LTDC_HandleTypeDef * hltdc)

Function description

Handle LTDC interrupt request.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_ConfigLayer

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

Configure the LTDC Layer according to the specified parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowSize

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowSize (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)

Function description

Set the LTDC window size.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize:** LTDC Pixel per line
- **YSize:** LTDC Line number
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowPosition

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowPosition (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)

Function description

Set the LTDC window position.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0:** LTDC window X offset
- **Y0:** LTDC window Y offset
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetPixelFormat

Function name

HAL_StatusTypeDef HAL_LTDC_SetPixelFormat (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)

Function description

Reconfigure the pixel format.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Pixelformat:** new pixel format value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetAlpha

Function name

HAL_StatusTypeDef HAL_LTDC_SetAlpha (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)

Function description

Reconfigure the layer alpha value.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha:** new alpha value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetAddress

Function name

HAL_StatusTypeDef HAL_LTDC_SetAddress (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)

Function description

Reconfigure the frame buffer Address.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address:** new address value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetPitch

Function name

HAL_StatusTypeDef HAL_LTDC_SetPitch (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)

Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx:** LTDC layer index concerned by the modification of line pitch.

Return values

- **HAL:** status

Notes

- This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above).

HAL_LTDC_ConfigColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)

Function description

Configure the color keying.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue:** the color key value
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_ConfigCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigCLUT (LTDC_HandleTypeDef * hltdc, const uint32_t * pCLUT, uint32_t CLUTSize, uint32_t LayerIdx)

Function description

Load the color lookup table.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pCLUT:** pointer to the color lookup table address.
- **CLUTSize:** the color lookup table size.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color keying.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableColorKeying

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color keying.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color lookup table.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableCLUT

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color lookup table.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_ProgramLineEvent

Function name

HAL_StatusTypeDef HAL_LTDC_ProgramLineEvent (LTDC_HandleTypeDef * hltdc, uint32_t Line)

Function description

Define the position of the line interrupt.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Line:** Line Interrupt Position.

Return values

- **HAL:** status

Notes

- User application may resort to HAL_LTDC_LineEventCallback() at line interrupt generation.

HAL_LTDC_EnableDither

Function name

HAL_StatusTypeDef HAL_LTDC_EnableDither (LTDC_HandleTypeDef * hltdc)

Function description

Enable Dither.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_DisableDither

Function name

HAL_StatusTypeDef HAL_LTDC_DisableDither (LTDC_HandleTypeDef * hltdc)

Function description

Disable Dither.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** status

HAL_LTDC_Reload

Function name

HAL_StatusTypeDef HAL_LTDC_Reload (LTDC_HandleTypeDef * hltdc, uint32_t ReloadType)

Function description

Reload LTDC Layers configuration.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **ReloadType:** This parameter can be one of the following values : LTDC_RELOAD_IMMEDIATE : Immediate Reload LTDC_RELOAD_VERTICAL_BLANKING : Reload in the next Vertical Blanking

Return values

- **HAL:** status

Notes

- User application may resort to HAL_LTDC_ReloadEventCallback() at reload interrupt generation.

HAL_LTDC_ConfigLayer_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigLayer_NoReload (LTDC_HandleTypeDef * hltdc, LTDC_LayerCfgTypeDef * pLayerCfg, uint32_t LayerIdx)

Function description

Configure the LTDC Layer according to the specified without reloading parameters in the LTDC_InitTypeDef and create the associated handle.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **pLayerCfg:** pointer to a LTDC_LayerCfgTypeDef structure that contains the configuration information for the Layer.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowSize_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowSize_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t XSize, uint32_t YSize, uint32_t LayerIdx)

Function description

Set the LTDC window size without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **XSize:** LTDC Pixel per line
- **YSize:** LTDC Line number
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetWindowPosition_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetWindowPosition_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t X0, uint32_t Y0, uint32_t LayerIdx)

Function description

Set the LTDC window position without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **X0:** LTDC window X offset
- **Y0:** LTDC window Y offset
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetPixelFormat_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetPixelFormat_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Pixelformat, uint32_t LayerIdx)

Function description

Reconfigure the pixel format without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDfef structure that contains the configuration information for the LTDC.
- **Pixelformat:** new pixel format value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetAlpha_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetAlpha_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Alpha, uint32_t LayerIdx)

Function description

Reconfigure the layer alpha value without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Alpha:** new alpha value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_SetAddress_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetAddress_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t Address, uint32_t LayerIdx)

Function description

Reconfigure the frame buffer Address without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **Address:** new address value.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return values

- **HAL:** status

HAL_LTDC_SetPitch_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_SetPitch_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LinePitchInPixels, uint32_t LayerIdx)

Function description

Function used to reconfigure the pitch for specific cases where the attached LayerIdx buffer have a width that is larger than the one intended to be displayed on screen.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LinePitchInPixels:** New line pitch in pixels to configure for LTDC layer 'LayerIdx'.
- **LayerIdx:** LTDC layer index concerned by the modification of line pitch.

Return values

- **HAL:** status

Notes

- This function should be called only after a previous call to HAL_LTDC_ConfigLayer() to modify the default pitch configured by HAL_LTDC_ConfigLayer() when required (refer to example described just above). Variant of the function HAL_LTDC_SetPitch without immediate reload.

HAL_LTDC_ConfigColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_ConfigColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t RGBValue, uint32_t LayerIdx)

Function description

Configure the color keying without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **RGBValue:** the color key value
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color keying without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableColorKeying_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableColorKeying_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color keying without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_EnableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_EnableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Enable the color lookup table without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_DisableCLUT_NoReload

Function name

HAL_StatusTypeDef HAL_LTDC_DisableCLUT_NoReload (LTDC_HandleTypeDef * hltdc, uint32_t LayerIdx)

Function description

Disable the color lookup table without reloading.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **LayerIdx:** LTDC Layer index. This parameter can be one of the following values: LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1)

Return values

- **HAL:** status

HAL_LTDC_GetState

Function name

HAL_LTDC_StateTypeDef HAL_LTDC_GetState (const LTDC_HandleTypeDef * hltdc)

Function description

Return the LTDC handle state.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **HAL:** state

HAL_LTDC_GetError

Function name

uint32_t HAL_LTDC_GetError (const LTDC_HandleTypeDef * hltdc)

Function description

Return the LTDC handle error code.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.

Return values

- **LTDC:** Error Code

46.3 LTDC Firmware driver defines

The following section lists the various define and macros of the module.

46.3.1 LTDC

LTDC

LTDC Alpha

LTDC_ALPHA

LTDC Constant Alpha mask

LTDC BACK COLOR

LTDC_COLOR

Color mask

LTDC Blending Factor1

LTDC_BLENDING_FACTOR1_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR1_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC Blending Factor2

LTDC_BLENDING_FACTOR2_CA

Blending factor : Cte Alpha

LTDC_BLENDING_FACTOR2_PAxCA

Blending factor : Cte Alpha x Pixel Alpha

LTDC DE POLARITY

LTDC_DEPOLARITY_AL

Data Enable, is active low.

LTDC_DEPOLARITY_AH

Data Enable, is active high.

LTDC Error Code

HAL_LTDC_ERROR_NONE

LTDC No error

HAL_LTDC_ERROR_TE

LTDC Transfer error

HAL_LTDC_ERROR_FU

LTDC FIFO Underrun

HAL_LTDC_ERROR_TIMEOUT

LTDC Timeout error

HAL_LTDC_ERROR_INVALID_CALLBACK

LTDC Invalid Callback error

LTDC Exported Macros

__HAL_LTDC_RESET_HANDLE_STATE

Description:

- Reset LTDC handle state.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None

__HAL_LTDC_ENABLE

Description:

- Enable the LTDC.

Parameters:

- __HANDLE__: LTDC handle

Return value:

- None.

__HAL_LTDC_DISABLE

Description:
- Disable the LTDC.

Parameters:
- __HANDLE__: LTDC handle

Return value:
- None.

__HAL_LTDC_LAYER_ENABLE

Description:
- Enable the LTDC Layer.

Parameters:
- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be enabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:
- None.

__HAL_LTDC_LAYER_DISABLE

Description:
- Disable the LTDC Layer.

Parameters:
- __HANDLE__: LTDC handle
- __LAYER__: Specify the layer to be disabled. This parameter can be LTDC_LAYER_1 (0) or LTDC_LAYER_2 (1).

Return value:
- None.

__HAL_LTDC_RELOAD_IMMEDIATE_CONFIG

Description:
- Reload immediately all LTDC Layers.

Parameters:
- __HANDLE__: LTDC handle

Return value:
- None.

__HAL_LTDC_VERTICAL_BLANKING_RELOAD_CONFIG

Description:
- Reload during vertical blanking period all LTDC Layers.

Parameters:
- __HANDLE__: LTDC handle

Return value:
- None.

__HAL_LTDC_GET_FLAG

Description:

- Get the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Get the specified flag. This parameter can be any combination of the following values:
  – LTDC_FLAG_LI: Line Interrupt flag
  – LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  – LTDC_FLAG_TE: Transfer Error interrupt flag
  – LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- The: state of FLAG (SET or RESET).

__HAL_LTDC_CLEAR_FLAG

Description:

- Clears the LTDC pending flags.

Parameters:

- __HANDLE__: LTDC handle
- __FLAG__: Specify the flag to clear. This parameter can be any combination of the following values:
  – LTDC_FLAG_LI: Line Interrupt flag
  – LTDC_FLAG_FU: FIFO Underrun Interrupt flag
  – LTDC_FLAG_TE: Transfer Error interrupt flag
  – LTDC_FLAG_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_ENABLE_IT

Description:

- Enables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be enabled. This parameter can be any combination of the following values:
  – LTDC_IT_LI: Line Interrupt flag
  – LTDC_IT_FU: FIFO Underrun Interrupt flag
  – LTDC_IT_TE: Transfer Error interrupt flag
  – LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_DISABLE_IT

Description:

- Disables the specified LTDC interrupts.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- None

__HAL_LTDC_GET_IT_SOURCE

Description:

- Check whether the specified LTDC interrupt has occurred or not.

Parameters:

- __HANDLE__: LTDC handle
- __INTERRUPT__: Specify the LTDC interrupt source to check. This parameter can be one of the following values:
  - LTDC_IT_LI: Line Interrupt flag
  - LTDC_IT_FU: FIFO Underrun Interrupt flag
  - LTDC_IT_TE: Transfer Error interrupt flag
  - LTDC_IT_RR: Register Reload Interrupt Flag

Return value:

- The: state of INTERRUPT (SET or RESET).

LTDC Exported Types

MAX_LAYER

LTDC Flags

LTDC_FLAG_LI

LTDC Line Interrupt Flag

LTDC_FLAG_FU

LTDC FIFO Underrun interrupt Flag

LTDC_FLAG_TE

LTDC Transfer Error interrupt Flag

LTDC_FLAG_RR

LTDC Register Reload interrupt Flag

LTDC HS POLARITY

LTDC_HSPOLARITY_AL

Horizontal Synchronization is active low.

LTDC_HSPOLARITY_AH

Horizontal Synchronization is active high.

LTDC Interrupts

LTDC_IT_LI

LTDC Line Interrupt

LTDC_IT_FU

LTDC FIFO Underrun Interrupt

LTDC_IT_TE

LTDC Transfer Error Interrupt

LTDC_IT_RR

LTDC Register Reload Interrupt

LTDC Layer

LTDC_LAYER_1

LTDC Layer 1

LTDC_LAYER_2

LTDC Layer 2

LTDC LAYER Config

LTDC_STOPPOSITION

LTDC Layer stop position

LTDC_STARTPOSITION

LTDC Layer start position

LTDC_COLOR_FRAME_BUFFER

LTDC Layer Line length

LTDC_LINE_NUMBER

LTDC Layer Line number

LTDC PC POLARITY

LTDC_PCPOLARITY_IPC

input pixel clock.

LTDC_PCPOLARITY_IIPC

inverted input pixel clock.

LTDC Pixel format

LTDC_PIXEL_FORMAT_ARGB8888

ARGB8888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB888

RGB888 LTDC pixel format

LTDC_PIXEL_FORMAT_RGB565

RGB565 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB1555

ARGB1555 LTDC pixel format

LTDC_PIXEL_FORMAT_ARGB4444

ARGB4444 LTDC pixel format

LTDC_PIXEL_FORMAT_L8

L8 LTDC pixel format

LTDC_PIXEL_FORMAT_AL44

AL44 LTDC pixel format

LTDC_PIXEL_FORMAT_AL88

AL88 LTDC pixel format

LTDC Reload Type

LTDC_RELOAD_IMMEDIATE

Immediate Reload

LTDC_RELOAD_VERTICAL_BLANKING

Vertical Blanking Reload

LTDC SYNC

LTDC_HORIZONTALSYNC

Horizontal synchronization width.

LTDC_VERTICALSYNC

Vertical synchronization height.

LTDC VS POLARITY

LTDC_VSPOLARITY_AL

Vertical Synchronization is active low.

LTDC_VSPOLARITY_AH

Vertical Synchronization is active high.

47 HAL LTDC Extension Driver

47.1 LTDCEx Firmware driver API description

The following section lists the various functions of the LTDCEx library.

47.1.1 Initialization and Configuration functions

This section provides functions allowing to:

- Initialize and configure the LTDC

This section contains the following APIs:

- *HAL_LTDCEx_StructInitFromVideoConfig()*
- *HAL_LTDCEx_StructInitFromAdaptedCommandConfig()*

47.1.2 Detailed description of functions

HAL_LTDCEx_StructInitFromVideoConfig

Function name

HAL_StatusTypeDef HAL_LTDCEx_StructInitFromVideoConfig (LTDC_HandleTypeDef * hltdc, DSI_VidCfgTypeDef * VidCfg)

Function description

Retrieve common parameters from DSI Video mode configuration structure.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **VidCfg:** pointer to a DSI_VidCfgTypeDef structure that contains the DSI video mode configuration parameters

Return values

- **HAL:** status

Notes

- The implementation of this function is taking into account the LTDC polarities inversion as described in the current LTDC specification

HAL_LTDCEx_StructInitFromAdaptedCommandConfig

Function name

HAL_StatusTypeDef HAL_LTDCEx_StructInitFromAdaptedCommandConfig (LTDC_HandleTypeDef * hltdc, DSI_CmdCfgTypeDef * CmdCfg)

Function description

Retrieve common parameters from DSI Adapted command mode configuration structure.

Parameters

- **hltdc:** pointer to a LTDC_HandleTypeDef structure that contains the configuration information for the LTDC.
- **CmdCfg:** pointer to a DSI_CmdCfgTypeDef structure that contains the DSI command mode configuration parameters

Return values

- **HAL:** status

Notes

- The implementation of this function is taking into account the LTDC polarities inversion as described in the current LTDC specification

48 HAL MDF Generic Driver

48.1 MDF Firmware driver registers structures

48.1.1 MDF_ClockTriggerTypeDef

MDF_ClockTriggerTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Source***
- ***uint32_t Edge***

Field Documentation

- ***FunctionalState MDF_ClockTriggerTypeDef::Activation***
  Output clock trigger enable/disable
- ***uint32_t MDF_ClockTriggerTypeDef::Source***
  Output clock trigger source. This parameter can be a value of ***MDF_ClockTriggerSource***
- ***uint32_t MDF_ClockTriggerTypeDef::Edge***
  Output clock trigger edge. This parameter can be a value of ***MDF_ClockTriggerEdge***

48.1.2 MDF_OutputClockTypeDef

MDF_OutputClockTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Pins***
- ***uint32_t Divider***
- ***MDF_ClockTriggerTypeDef Trigger***

Field Documentation

- ***FunctionalState MDF_OutputClockTypeDef::Activation***
  Output clock enable/disable
- ***uint32_t MDF_OutputClockTypeDef::Pins***
  Output clock pins. This parameter can be a value of ***MDF_OuputClockPins***
- ***uint32_t MDF_OutputClockTypeDef::Divider***
  Output clock divider. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***MDF_ClockTriggerTypeDef MDF_OutputClockTypeDef::Trigger***
  Output clock trigger parameters

48.1.3 MDF_CommonParamTypeDef

MDF_CommonParamTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***uint32_t InterleavedFilters***
- ***uint32_t ProcClockDivider***
- ***MDF_OutputClockTypeDef OutputClock***

Field Documentation

- ***uint32_t MDF_CommonParamTypeDef::InterleavedFilters***
  Number of filters in interleaved mode with filter 0. This parameter must be a number between Min_Data = 0 and Max_Data = 1 for STM32U535xx/STM32U545xx devices. This parameter must be a number between Min_Data = 0 and Max_Data = 5 for other devices.
  Note:
  – This parameter is not used for ADF instance
- ***uint32_t MDF_CommonParamTypeDef::ProcClockDivider***
  Processing clock divider. This parameter must be a number between Min_Data = 1 and Max_Data = 128

- ***MDF_OutputClockTypeDef MDF_CommonParamTypeDef::OutputClock***
  Output clock parameters

48.1.4 MDF_SerialInterfaceTypeDef

MDF_SerialInterfaceTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Mode***
- ***uint32_t ClockSource***
- ***uint32_t Threshold***

Field Documentation

- ***FunctionalState MDF_SerialInterfaceTypeDef::Activation***
  Serial interface enable/disable
- ***uint32_t MDF_SerialInterfaceTypeDef::Mode***
  Serial interface mode. This parameter can be a value of ***MDF_SitfMode***
- ***uint32_t MDF_SerialInterfaceTypeDef::ClockSource***
  Serial interface clock source. This parameter can be a value of ***MDF_SitfClockSource***
- ***uint32_t MDF_SerialInterfaceTypeDef::Threshold***
  SPI threshold for clock absence detection or Manchester symbol threshold. This parameter must be a number between Min_Data = 4 and Max_Data = 31

48.1.5 MDF_InitTypeDef

MDF_InitTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***MDF_CommonParamTypeDef CommonParam***
- ***MDF_SerialInterfaceTypeDef SerialInterface***
- ***uint32_t FilterBistream***

Field Documentation

- ***MDF_CommonParamTypeDef MDF_InitTypeDef::CommonParam***
  MDF common parameters
- ***MDF_SerialInterfaceTypeDef MDF_InitTypeDef::SerialInterface***
  MDF serial interface parameters
- ***uint32_t MDF_InitTypeDef::FilterBistream***
  MDF filter bitstream selection. This parameter can be a value of ***MDF_FilterBitstream***

48.1.6 __MDF_HandleTypeDef

__MDF_HandleTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***MDF_Filter_TypeDef * Instance***
- ***MDF_InitTypeDef Init***
- ***DMA_HandleTypeDef * hdma***
- ***__IO HAL_MDF_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* OldCallback***
- ***void(* AcqCpltCallback***
- ***void(* AcqHalfCpltCallback***
- ***void(* SndLvCallback***
- ***void(* SadCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***MDF_Filter_TypeDef* __MDF_HandleTypeDef::Instance***
  MDF instance
- ***MDF_InitTypeDef __MDF_HandleTypeDef::Init***
  MDF init parameters
- ***DMA_HandleTypeDef* __MDF_HandleTypeDef::hdma***
  Pointer on DMA handler for acquisitions
- ***__IO HAL_MDF_StateTypeDef __MDF_HandleTypeDef::State***
  MDF state
- ***__IO uint32_t __MDF_HandleTypeDef::ErrorCode***
  MDF error code
- ***void(* __MDF_HandleTypeDef::OldCallback)(struct __MDF_HandleTypeDef *hmdf, uint32_t Threshold)***
  MDF out-off limit detector callback.
  Note:
  – Not used for ADF instance
- ***void(* __MDF_HandleTypeDef::AcqCpltCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF acquisition complete callback
- ***void(* __MDF_HandleTypeDef::AcqHalfCpltCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF acquisition half complete callback
- ***void(* __MDF_HandleTypeDef::SndLvCallback)(struct __MDF_HandleTypeDef *hmdf, uint32_t SoundLevel, uint32_t AmbientNoise)***
  MDF sound level callback.
  Note:
  – Not used for MDF instance
- ***void(* __MDF_HandleTypeDef::SadCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF sound activity detector callback.
  Note:
  – Not used for MDF instance
- ***void(* __MDF_HandleTypeDef::ErrorCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF error callback
- ***void(* __MDF_HandleTypeDef::MspInitCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF MSP init callback
- ***void(* __MDF_HandleTypeDef::MspDeInitCallback)(struct __MDF_HandleTypeDef *hmdf)***
  MDF MSP de-init callback

48.1.7 MDF_ReshapeFilterTypeDef

MDF_ReshapeFilterTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t DecimationRatio***

Field Documentation

- ***FunctionalState MDF_ReshapeFilterTypeDef::Activation***
  Reshape filter enable/disable
- ***uint32_t MDF_ReshapeFilterTypeDef::DecimationRatio***
  Reshape filter decimation ratio. This parameter can be a value of ***MDF_ReshapeDecimationRatio***

48.1.8 MDF_HighPassFilterTypeDef

MDF_HighPassFilterTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t CutOffFrequency***

Field Documentation

- ***FunctionalState MDF_HighPassFilterTypeDef::Activation***
  High pass filter enable/disable

- ***uint32_t MDF_HighPassFilterTypeDef::CutOffFrequency***
  High pass filter cut-off frequency. This parameter can be a value of ***MDF_HighPassCutOffFreq***

48.1.9 MDF_IntegratorTypeDef

MDF_IntegratorTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Value***
- ***uint32_t OutputDivision***

Field Documentation

- ***FunctionalState MDF_IntegratorTypeDef::Activation***
  Integrator enable/disable
- ***uint32_t MDF_IntegratorTypeDef::Value***
  Integrator value. This parameter must be a number between Min_Data = 2 and Max_Data = 128
- ***uint32_t MDF_IntegratorTypeDef::OutputDivision***
  Integrator output division. This parameter can be a value of ***MDF_IntegratorOutputDiv***

48.1.10 MDF_SoundActivityTypeDef

MDF_SoundActivityTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t Mode***
- ***uint32_t FrameSize***
- ***FunctionalState Hysteresis***
- ***uint32_t SoundTriggerEvent***
- ***uint32_t DataMemoryTransfer***
- ***uint32_t MinNoiseLevel***
- ***uint32_t HangoverWindow***
- ***uint32_t LearningFrames***
- ***uint32_t AmbientNoiseSlope***
- ***uint32_t SignalNoiseThreshold***
- ***FunctionalState SoundLevelInterrupt***

Field Documentation

- ***FunctionalState MDF_SoundActivityTypeDef::Activation***
  Sound activity detector enable/disable
- ***uint32_t MDF_SoundActivityTypeDef::Mode***
  Sound activity detector mode. This parameter can be a value of ***MDF_SadMode***
- ***uint32_t MDF_SoundActivityTypeDef::FrameSize***
  Size of one frame to compute short-term signal level. This parameter can be a value of ***MDF_SadFrameSize***
- ***FunctionalState MDF_SoundActivityTypeDef::Hysteresis***
  Hysteresis enable/disable.
  Note:
  – This parameter is not used if Mode is set to MDF_SAD_AMBIENT_NOISE_ESTIMATOR
- ***uint32_t MDF_SoundActivityTypeDef::SoundTriggerEvent***
  Sound trigger event configuration. This parameter can be a value of ***MDF_SadSoundTriggerEvent***
- ***uint32_t MDF_SoundActivityTypeDef::DataMemoryTransfer***
  Data memory transfer mode. This parameter can be a value of ***MDF_SadDataMemoryTransfer***
- ***uint32_t MDF_SoundActivityTypeDef::MinNoiseLevel***
  Minimum noise level. This parameter must be a number between Min_Data = 0 and Max_Data = 8191
- ***uint32_t MDF_SoundActivityTypeDef::HangoverWindow***
  Hangover time window in frames. This parameter can be a value of ***MDF_SadHangoverWindow***

- ***uint32_t MDF_SoundActivityTypeDef::LearningFrames***
  Number of learning frames for the first estimation of noise level. This parameter can be a value of ***MDF_SadLearningFrames***
- ***uint32_t MDF_SoundActivityTypeDef::AmbientNoiseSlope***
  Ambient noise slope control. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
  Note:
  – This parameter is not used if Mode is set to MDF_SAD_SOUND_DETECTOR
- ***uint32_t MDF_SoundActivityTypeDef::SignalNoiseThreshold***
  Signal to noise threshold. This parameter can be a value of ***MDF_SadSignalNoiseThreshold***
- ***FunctionalState MDF_SoundActivityTypeDef::SoundLevelInterrupt***
  Sound level interrupt enable/disable.
  Note:
  – This interrupt is mainly used for debug purpose

48.1.11 MDF_FilterTriggerTypeDef

MDF_FilterTriggerTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Edge***

Field Documentation

- ***uint32_t MDF_FilterTriggerTypeDef::Source***
  Filter trigger source. This parameter can be a value of ***MDF_FilterTriggerSource***
- ***uint32_t MDF_FilterTriggerTypeDef::Edge***
  Filter trigger edge. This parameter can be a value of ***MDF_FilterTriggerEdge***

48.1.12 MDF_FilterConfigTypeDef

MDF_FilterConfigTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***uint32_t DataSource***
- ***uint32_t Delay***
- ***uint32_t CicMode***
- ***uint32_t DecimationRatio***
- ***int32_t Offset***
- ***int32_t Gain***
- ***MDF_ReshapeFilterTypeDef ReshapeFilter***
- ***MDF_HighPassFilterTypeDef HighPassFilter***
- ***MDF_IntegratorTypeDef Integrator***
- ***MDF_SoundActivityTypeDef SoundActivity***
- ***uint32_t AcquisitionMode***
- ***uint32_t FifoThreshold***
- ***uint32_t DiscardSamples***
- ***MDF_FilterTriggerTypeDef Trigger***
- ***uint32_t SnapshotFormat***

Field Documentation

- ***uint32_t MDF_FilterConfigTypeDef::DataSource***
  Filter data source. This parameter can be a value of ***MDF_DataSource***
- ***uint32_t MDF_FilterConfigTypeDef::Delay***
  Delay to apply on data source in number of samples. This parameter must be a number between Min_Data = 0 and Max_Data = 127
- ***uint32_t MDF_FilterConfigTypeDef::CicMode***
  CIC filter mode. This parameter can be a value of ***MDF_CicMode***

- ***uint32_t MDF_FilterConfigTypeDef::DecimationRatio***
  Filter decimation ratio. This parameter must be a number between Min_Data = 2 and Max_Data = 512
- ***int32_t MDF_FilterConfigTypeDef::Offset***
  Filter offset error compensation. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431.
  Note:
  – This parameter is not used for ADF instance
- ***int32_t MDF_FilterConfigTypeDef::Gain***
  Filter gain in step of around 3db (from -48db to 72dB). This parameter must be a number between Min_Data = -16 and Max_Data = 24
- ***MDF_ReshapeFilterTypeDef MDF_FilterConfigTypeDef::ReshapeFilter***
  Reshape filter configuration
- ***MDF_HighPassFilterTypeDef MDF_FilterConfigTypeDef::HighPassFilter***
  High pass filter configuration
- ***MDF_IntegratorTypeDef MDF_FilterConfigTypeDef::Integrator***
  Integrator configuration.
  Note:
  – This parameter is not used for ADF instance
- ***MDF_SoundActivityTypeDef MDF_FilterConfigTypeDef::SoundActivity***
  Sound activity detector configuration.
  Note:
  – This parameter is not used for MDF instance
- ***uint32_t MDF_FilterConfigTypeDef::AcquisitionMode***
  Filter acquisition mode. This parameter can be a value of ***MDF_AcquisitionMode***
- ***uint32_t MDF_FilterConfigTypeDef::FifoThreshold***
  Filter RXFIFO threshold. This parameter can be a value of ***MDF_FifoThreshold***
- ***uint32_t MDF_FilterConfigTypeDef::DiscardSamples***
  Number of samples to discard after filter enable. This parameter must be a number between Min_Data = 0 and Max_Data = 255
- ***MDF_FilterTriggerTypeDef MDF_FilterConfigTypeDef::Trigger***
  Filter trigger configuration.
  Note:
  – This parameter is not used if AcquisitionMode is set to MDF_MODE_ASYNC_CONT or MDF_MODE_ASYNC_SINGLE
- ***uint32_t MDF_FilterConfigTypeDef::SnapshotFormat***
  Snapshot format. This parameter can be a value of ***MDF_SnapshotFormat***.
  Note:
  – This parameter is used only if AcquisitionMode is set to MDF_MODE_SYNC_SNAPSHOT and for MDF instance

48.1.13 MDF_SnapshotParamTypeDef

MDF_SnapshotParamTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***int32_t Value***
- ***uint32_t DecimationCounter***
- ***uint32_t IntegratorCounter***

Field Documentation

- ***int32_t MDF_SnapshotParamTypeDef::Value***
  Snapshot acquisition value on 16 or 23 MSB depending on snapshot format
- ***uint32_t MDF_SnapshotParamTypeDef::DecimationCounter***
  Decimation counter value when snapshot trigger occurs
- ***uint32_t MDF_SnapshotParamTypeDef::IntegratorCounter***
  Integrator counter value when snapshot trigger occurs. This value is available only if snapshot format is 16 bits resolution

48.1.14 MDF_DmaConfigTypeDef

MDF_DmaConfigTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***uint32_t Address***
- ***uint32_t DataLength***
- ***FunctionalState MsbOnly***

Field Documentation

- ***uint32_t MDF_DmaConfigTypeDef::Address***
  DMA destination address
- ***uint32_t MDF_DmaConfigTypeDef::DataLength***
  Length of data to transfer in bytes
- ***FunctionalState MDF_DmaConfigTypeDef::MsbOnly***
  Transfer only the 16MSB of the acquistion data

48.1.15 MDF_ScdConfigTypeDef

MDF_ScdConfigTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***uint32_t Threshold***
- ***uint32_t BreakSignal***

Field Documentation

- ***uint32_t MDF_ScdConfigTypeDef::Threshold***
  Short-circuit detector threshold. This parameter must be a number between Min_Data = 2 and Max_Data = 256
- ***uint32_t MDF_ScdConfigTypeDef::BreakSignal***
  Break signal assignment for short-circuit detector. This parameter can be a values combination of ***MDF_BreakSignals***

48.1.16 MDF_OldConfigTypeDef

MDF_OldConfigTypeDef is defined in the stm32u5xx_hal_mdf.h

Data Fields

- ***uint32_t OldCicMode***
- ***uint32_t OldDecimationRatio***
- ***int32_t HighThreshold***
- ***int32_t LowThreshold***
- ***uint32_t OldEventConfig***
- ***uint32_t BreakSignal***

Field Documentation

- ***uint32_t MDF_OldConfigTypeDef::OldCicMode***
  Out-off limit detector CIC filter mode. This parameter can be a value of ***MDF_OldCicMode***
- ***uint32_t MDF_OldConfigTypeDef::OldDecimationRatio***
  Out-off limit detector decimation ratio. This parameter must be a number between Min_Data = 1 and Max_Data = 32
- ***int32_t MDF_OldConfigTypeDef::HighThreshold***
  Out-off limit detector high threshold. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431
- ***int32_t MDF_OldConfigTypeDef::LowThreshold***
  Out-off limit detector low threshold. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431
- ***uint32_t MDF_OldConfigTypeDef::OldEventConfig***
  Out-off limit event configuration. This parameter can be a value of ***MDF_OldEventConfig***
- ***uint32_t MDF_OldConfigTypeDef::BreakSignal***
  Break signal assignment for out-off limit detector. This parameter can be a values combination of ***MDF_BreakSignals***

48.2 MDF Firmware driver API description

The following section lists the various functions of the MDF library.

48.2.1 How to use this driver

Initialization and de-initialization

1. User has first to initialize MDF or ADF instance.
2. As prerequisite, fill in the HAL_MDF_MspInit() :
   - Enable MDFz or ADFz clock interface with __HAL_RCC_MDFz_CLK_ENABLE() or __HAL_RCC_ADFz_CLK_ENABLE().
   - Enable the clocks for the used GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
   - Configure these pins in alternate mode using HAL_GPIO_Init().
   - If interrupt mode is used, enable and configure MDFz_FLTx or ADFz interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
   - If DMA mode is used, initialize and configure DMA.
3. Configure the common parameters (only for first MDF or ADF instance init), serial interface parameters and filter bitstream selection by calling HAL_MDF_Init() function.

1. User can de-initialize MDF or ADF instance with HAL_MDF_DeInit() function.

Acquisition

1. Configure filter parameters and start acquisition using HAL_MDF_AcqStart(), HAL_MDF_AcqStart_IT() or HAL_MDF_AcqStart_DMA().
2. In polling mode :
   - Use HAL_MDF_PollForAcq() to detect the end of acquisition. Use HAL_MDF_GetAcqValue to get acquisition value.
   - Only for MDF instance, use HAL_MDF_PollForSnapshotAcq() to detect the end of snapshot acquisition. Use HAL_MDF_GetSnapshotAcqValue to get snapshot acquisition value.
   - Only for ADF instance, use HAL_MDF_PollForSndLvl() to detect and get new sound level value and ambient noise value.
   - Only for ADF instance, use HAL_MDF_PollForSad() to detect sound activity.
3. In interrupt mode :
   - HAL_MDF_AcqCpltCallback() will be called at the end of acquisition. Use HAL_MDF_GetAcqValue to get acquisition value or use HAL_MDF_GetSnapshotAcqValue to get snapshot acquisition value.
   - Only for ADF instance, HAL_MDF_SndLvlCallback() will be called when new sound level and ambient noise values are available.
   - Only for ADF instance, HAL_MDF_SadCallback() will be called when sound activity detection occurs.
   - HAL_MDF_ErrorCallback() will be called if overflow, filter overrun or saturation occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
4. In DMA mode :
   - HAL_MDF_AcqHalfCpltCallback() and HAL_MDF_AcqCpltCallback() will be called respectively at the half acquisition and at the acquisition complete.
   - Only for ADF instance, HAL_MDF_SndLvlCallback() will be called when new sound level and ambient noise values are available.
   - Only for ADF instance, HAL_MDF_SadCallback() will be called when sound activity detection occurs.
   - HAL_MDF_ErrorCallback() will be called if overflow, filter overrun, saturation or DMA error occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
5. Use HAL_MDF_GenerateTrgo() to generate pulse on TRGO signal.
6. During acquisition, use HAL_MDF_SetDelay() and HAL_MDF_GetDelay() to respectively set and get the delay on data source.
7. During acquisition, use HAL_MDF_SetGain() and HAL_MDF_GetGain() to respectively set and get the filter gain.

8. During acquisition, use HAL_MDF_SetOffset() and HAL_MDF_GetOffset() to respectively set and get the filter offset error compensation.
9. During acquisition, only for MDF instance, use HAL_MDF_SetOffset() and HAL_MDF_GetOffset() to respectively set and get the filter offset error compensation.
10. Stop acquisition using HAL_MDF_AcqStop(), HAL_MDF_AcqStop_IT() or HAL_MDF_AcqStop_DMA().

Clock absence detection

1. Clock absence detection is always enabled so no need to start clock absence detection in polling mode. Use HAL_MDF_CkabStart_IT() to start clock absence detection in interrupt mode.
2. In polling mode, use HAL_MDF_PollForCkab() to detect the clock absence.
3. In interrupt mode, HAL_MDF_ErrorCallback() will be called if clock absence detection occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
4. Stop clock absence detection in interrupt mode using HAL_MDF_CkabStop_IT().

Short circuit detection

1. Only for MDF instance, start short circuit detection using HAL_MDF_ScdStart() or HAL_MDF_ScdStart_IT().
2. In polling mode, use HAL_MDF_PollForScd() to detect short circuit.
3. In interrupt mode, HAL_MDF_ErrorCallback() will be called if short circuit detection occurs. Use HAL_MDF_GetErrorCode() to get the corresponding error.
4. Stop short circuit detection using HAL_MDF_ScdStop() or HAL_MDF_ScdStop_IT().

Out-off limit detection

1. Only for MDF instance, start out-off limit detection using HAL_MDF_OldStart() or HAL_MDF_OldStart_IT().
2. In polling mode, use HAL_MDF_PollForOld() to detect out-off limit and to get threshold information.
3. In interrupt mode, HAL_MDF_OldCallback() will be called if out-off limit detection occurs.
4. Stop out-off limit detection using HAL_MDF_OldStop() or HAL_MDF_OldStop_IT().

generic functions

1. HAL_MDF_IRQHandler will be called when MDF or ADF interrupt occurs.
2. HAL_MDF_ErrorCallback will be called when MDF or ADF error occurs.
3. Use HAL_MDF_GetState() to get the current MDF or ADF instance state.
4. Use HAL_MDF_GetErrorCode() to get the current MDF or ADF instance error code.

Callback registration

The compilation define USE_HAL_MDF_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_MDF_RegisterCallback(), HAL_MDF_RegisterOldCallback() or HAL_MDF_RegisterSndLvlCallback() to register a user callback.

Function HAL_MDF_RegisterCallback() allows to register following callbacks :

- AcqCpltCallback : Acquisition complete callback.
- AcqHalfCpltCallback : Acquisition half complete callback.
- SadCallback : Sound activity detection callback (only for ADF instance).
- ErrorCallback : Error callback.
- MspInitCallback : MSP init callback.
- MspDeInitCallback : MSP de-init callback.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For MDF instance and for specific out-off limit detection callback use dedicated register callback : HAL_MDF_RegisterOldCallback().

For ADF instance and for specific sound level callback use dedicated register callback : HAL_MDF_RegisterSndLvlCallback().

Use function HAL_MDF_UnRegisterCallback() to reset a callback to the default weak function.

HAL_MDF_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID.

This function allows to reset following callbacks :

- AcqCpltCallback : Acquisition complete callback.
- AcqHalfCpltCallback : Acquisition half complete callback.
- SadCallback : Sound activity detection callback (only for ADF instance).
- ErrorCallback : Error callback.
- MspInitCallback : MSP init callback.
- MspDeInitCallback : MSP de-init callback.

For MDF instance and for specific out-off limit detection callback use dedicated unregister callback : HAL_MDF_UnRegisterOldCallback().

For ADF instance and for specific sound level callback use dedicated unregister callback : HAL_MDF_UnRegisterSndLvlCallback().

By default, after the call of init function and if the state is RESET all callbacks are reset to the corresponding legacy weak functions : examples HAL_MDF_AcqCpltCallback(), HAL_MDF_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the init and de-init only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the init and de-init keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the init/de-init. In that case first register the MspInit/MspDeInit user callbacks using HAL_MDF_RegisterCallback() before calling init or de-init function.

When the compilation define USE_HAL_MDF_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

48.2.2 Initialization and de-initialization functions

This section provides functions allowing to :

- Initialize the MDF or ADF instance.
- De-initialize the MDF or ADF instance.
- Register and unregister callbacks.

This section contains the following APIs:

- *HAL_MDF_Init()*
- *HAL_MDF_DeInit()*
- *HAL_MDF_MspInit()*
- *HAL_MDF_MspDeInit()*
- *HAL_MDF_RegisterCallback()*
- *HAL_MDF_UnRegisterCallback()*
- *HAL_MDF_RegisterOldCallback()*
- *HAL_MDF_UnRegisterOldCallback()*
- *HAL_MDF_RegisterSndLvlCallback()*
- *HAL_MDF_UnRegisterSndLvlCallback()*

48.2.3 Acquisition functions

This section provides functions allowing to :

- Start and stop acquisition in polling, interrupt or DMA mode.
- Wait and get acquisition values.
- Generate pulse on TRGO signal.
- Modify and get some filter parameters during acquisition.
- Wait and get sound level values for ADF instance.
- Detect sound activity for ADF instance.

This section contains the following APIs:

- *HAL_MDF_AcqStart()*
- *HAL_MDF_PollForAcq()*

- *HAL_MDF_PollForSnapshotAcq()*
- *HAL_MDF_GetAcqValue()*
- *HAL_MDF_GetSnapshotAcqValue()*
- *HAL_MDF_AcqStop()*
- *HAL_MDF_AcqStart_IT()*
- *HAL_MDF_AcqStop_IT()*
- *HAL_MDF_AcqStart_DMA()*
- *HAL_MDF_AcqStop_DMA()*
- *HAL_MDF_GenerateTrgo()*
- *HAL_MDF_SetDelay()*
- *HAL_MDF_GetDelay()*
- *HAL_MDF_SetGain()*
- *HAL_MDF_GetGain()*
- *HAL_MDF_SetOffset()*
- *HAL_MDF_GetOffset()*
- *HAL_MDF_PollForSndLvl()*
- *HAL_MDF_PollForSad()*
- *HAL_MDF_AcqCpltCallback()*
- *HAL_MDF_AcqHalfCpltCallback()*
- *HAL_MDF_SndLvlCallback()*
- *HAL_MDF_SadCallback()*

48.2.4 Clock absence detection functions

This section provides functions allowing to :

- Start and stop clock absence detection in interrupt mode.
- Detect clock absence.

This section contains the following APIs:

- *HAL_MDF_PollForCkab()*
- *HAL_MDF_CkabStart_IT()*
- *HAL_MDF_CkabStop_IT()*

48.2.5 Short circuit detection functions

This section provides functions available only for MDF instance allowing to :

- Start and stop short circuit detection in polling and interrupt mode.
- Detect short circuit.

This section contains the following APIs:

- *HAL_MDF_ScdStart()*
- *HAL_MDF_PollForScd()*
- *HAL_MDF_ScdStop()*
- *HAL_MDF_ScdStart_IT()*
- *HAL_MDF_ScdStop_IT()*

48.2.6 Out-off limit detection functions

This section provides functions available only for MDF instance allowing to :

- Start and stop out-off limit detection in polling and interrupt mode.
- Detect short circuit and get threshold information.

This section contains the following APIs:

- *HAL_MDF_OldStart()*
- *HAL_MDF_PollForOld()*

- *HAL_MDF_OldStop()*
- *HAL_MDF_OldStart_IT()*
- *HAL_MDF_OldStop_IT()*
- *HAL_MDF_OldCallback()*

48.2.7 Generic functions

This section provides functions allowing to :

- Handle MDF interrupt.
- Inform user that error occurs.
- Get the current MDF instance state.
- Get the current MDF instance error code.

This section contains the following APIs:

- *HAL_MDF_IRQHandler()*
- *HAL_MDF_ErrorCallback()*
- *HAL_MDF_GetState()*
- *HAL_MDF_GetError()*

48.2.8 Detailed description of functions

HAL_MDF_Init

Function name

HAL_StatusTypeDef HAL_MDF_Init (MDF_HandleTypeDef * hmdf)

Function description

Initialize the MDF instance according to the specified parameters in the MDF_InitTypeDef structure and initialize the associated handle.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_DeInit

Function name

HAL_StatusTypeDef HAL_MDF_DeInit (MDF_HandleTypeDef * hmdf)

Function description

De-initialize the MDF instance.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_MspInit

Function name

void HAL_MDF_MspInit (MDF_HandleTypeDef * hmdf)

Function description

Initialize the MDF instance MSP.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_MspDeInit

Function name

void HAL_MDF_MspDeInit (MDF_HandleTypeDef * hmdf)

Function description

De-initialize the MDF instance MSP.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MDF_RegisterCallback (MDF_HandleTypeDef * hmdf, HAL_MDF_CallbackIDTypeDef CallbackID, pMDF_CallbackTypeDef pCallback)

Function description

Register a user MDF callback to be used instead of the weak predefined callback.

Parameters

- **hmdf:** MDF handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  - HAL_MDF_ACQ_COMPLETE_CB_ID acquisition complete callback ID.
  - HAL_MDF_ACQ_HALFCOMPLETE_CB_ID acquisition half complete callback ID.
  - HAL_MDF_SAD_CB_ID sound activity detector callback ID (only for ADF instance).
  - HAL_MDF_ERROR_CB_ID error callback ID.
  - HAL_MDF_MSPINIT_CB_ID MSP init callback ID.
  - HAL_MDF_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_MDF_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MDF_UnRegisterCallback (MDF_HandleTypeDef * hmdf, HAL_MDF_CallbackIDTypeDef CallbackID)

Function description

Unregister a user MDF callback.

Parameters

- **hmdf:** MDF handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  - HAL_MDF_ACQ_COMPLETE_CB_ID acquisition complete callback ID.
  - HAL_MDF_ACQ_HALFCOMPLETE_CB_ID acquisition half complete callback ID.
  - HAL_MDF_SAD_CB_ID sound activity detector callback ID (only for ADF instance).
  - HAL_MDF_ERROR_CB_ID error callback ID.
  - HAL_MDF_MSPINIT_CB_ID MSP init callback ID.
  - HAL_MDF_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_MDF_RegisterOldCallback

Function name

HAL_StatusTypeDef HAL_MDF_RegisterOldCallback (MDF_HandleTypeDef * hmdf, pMDF_OldCallbackTypeDef pCallback)

Function description

Register specific MDF out-off limit detector callback to be used instead of the weak predefined callback.

Parameters

- **hmdf:** MDF handle.
- **pCallback:** pointer to the out-off limit detector callback function.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_UnRegisterOldCallback

Function name

HAL_StatusTypeDef HAL_MDF_UnRegisterOldCallback (MDF_HandleTypeDef * hmdf)

Function description

Unregister the specific MDF out-off limit detector callback.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_RegisterSndLvlCallback

Function name

HAL_StatusTypeDef HAL_MDF_RegisterSndLvlCallback (MDF_HandleTypeDef * hmdf, pMDF_SndLvlCallbackTypeDef pCallback)

Function description

Register specific MDF sound level callback to be used instead of the weak predefined callback.

Parameters

- **hmdf:** MDF handle.
- **pCallback:** pointer to the sound level callback function.

Return values

- **HAL:** status.

Notes

- This function must not be used with MDF instance.

HAL_MDF_UnRegisterSndLvlCallback

Function name

HAL_StatusTypeDef HAL_MDF_UnRegisterSndLvlCallback (MDF_HandleTypeDef * hmdf)

Function description

Unregister the specific MDF sound level callback.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

Notes

- This function must not be used with MDF instance.

HAL_MDF_AcqStart

Function name

HAL_StatusTypeDef HAL_MDF_AcqStart (MDF_HandleTypeDef * hmdf, const MDF_FilterConfigTypeDef * pFilterConfig)

Function description

This function allows to start acquisition in polling mode.

Parameters

- **hmdf:** MDF handle.
- **pFilterConfig:** Filter configuration parameters.

Return values

- **HAL:** status.

HAL_MDF_PollForAcq

Function name

HAL_StatusTypeDef HAL_MDF_PollForAcq (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

This function allows to poll for available acquisition value.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status.

HAL_MDF_PollForSnapshotAcq

Function name

HAL_StatusTypeDef HAL_MDF_PollForSnapshotAcq (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

This function allows to poll for available snapshot acquisition value.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_GetAcqValue

Function name

HAL_StatusTypeDef HAL_MDF_GetAcqValue (const MDF_HandleTypeDef * hmdf, int32_t * pValue)

Function description

This function allows to get acquisition value.

Parameters

- **hmdf:** MDF handle.
- **pValue:** Acquisition value on 24 MSB.

Return values

- **HAL:** status.

HAL_MDF_GetSnapshotAcqValue

Function name

HAL_StatusTypeDef HAL_MDF_GetSnapshotAcqValue (MDF_HandleTypeDef * hmdf, MDF_SnapshotParamTypeDef * pSnapshotParam)

Function description

This function allows to get snapshot acquisition value.

Parameters

- **hmdf:** MDF handle.
- **pSnapshotParam:** Snapshot parameters.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_AcqStop

Function name

HAL_StatusTypeDef HAL_MDF_AcqStop (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop acquisition in polling mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_AcqStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_AcqStart_IT (MDF_HandleTypeDef * hmdf, const MDF_FilterConfigTypeDef * pFilterConfig)

Function description

This function allows to start acquisition in interrupt mode.

Parameters

- **hmdf:** MDF handle.
- **pFilterConfig:** Filter configuration parameters.

Return values

- **HAL:** status.

HAL_MDF_AcqStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_AcqStop_IT (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop acquisition in interrupt mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_AcqStart_DMA

Function name

HAL_StatusTypeDef HAL_MDF_AcqStart_DMA (MDF_HandleTypeDef * hmdf, const MDF_FilterConfigTypeDef * pFilterConfig, const MDF_DmaConfigTypeDef * pDmaConfig)

Function description

This function allows to start acquisition in DMA mode.

Parameters

- **hmdf:** MDF handle.
- **pFilterConfig:** Filter configuration parameters.
- **pDmaConfig:** DMA configuration parameters.

Return values

- **HAL:** status.

HAL_MDF_AcqStop_DMA

Function name

HAL_StatusTypeDef HAL_MDF_AcqStop_DMA (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop acquisition in DMA mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_GenerateTrgo

Function name

HAL_StatusTypeDef HAL_MDF_GenerateTrgo (const MDF_HandleTypeDef * hmdf)

Function description

This function allows to generate pulse on TRGO signal.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_SetDelay

Function name

HAL_StatusTypeDef HAL_MDF_SetDelay (MDF_HandleTypeDef * hmdf, uint32_t Delay)

Function description

This function allows to set delay to apply on data source in number of samples.

Parameters

- **hmdf:** MDF handle.
- **Delay:** Delay to apply on data source in number of samples. This parameter must be a number between Min_Data = 0 and Max_Data = 127.

Return values

- **HAL:** status.

HAL_MDF_GetDelay

Function name

HAL_StatusTypeDef HAL_MDF_GetDelay (const MDF_HandleTypeDef * hmdf, uint32_t * pDelay)

Function description

This function allows to get current delay applied on data source in number of samples.

Parameters

- **hmdf:** MDF handle.
- **pDelay:** Current delay applied on data source in number of samples. This value is between Min_Data = 0 and Max_Data = 127.

Return values

- **HAL:** status.

HAL_MDF_SetGain

Function name

HAL_StatusTypeDef HAL_MDF_SetGain (MDF_HandleTypeDef * hmdf, int32_t Gain)

Function description

This function allows to set filter gain.

Parameters

- **hmdf:** MDF handle.
- **Gain:** Filter gain in step of around 3db (from -48db to 72dB). This parameter must be a number between Min_Data = -16 and Max_Data = 24.

Return values

- **HAL:** status.

HAL_MDF_GetGain

Function name

HAL_StatusTypeDef HAL_MDF_GetGain (const MDF_HandleTypeDef * hmdf, int32_t * pGain)

Function description

This function allows to get filter gain.

Parameters

- **hmdf:** MDF handle.
- **pGain:** Filter gain in step of around 3db (from -48db to 72dB). This parameter is between Min_Data = -16 and Max_Data = 24.

Return values

- **HAL:** status.

HAL_MDF_SetOffset

Function name

HAL_StatusTypeDef HAL_MDF_SetOffset (MDF_HandleTypeDef * hmdf, int32_t Offset)

Function description

This function allows to set filter offset error compensation.

Parameters

- **hmdf:** MDF handle.
- **Offset:** Filter offset error compensation. This parameter must be a number between Min_Data = -33554432 and Max_Data = 33554431.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_GetOffset

Function name

HAL_StatusTypeDef HAL_MDF_GetOffset (const MDF_HandleTypeDef * hmdf, int32_t * pOffset)

Function description

This function allows to get filter offset error compensation.

Parameters

- **hmdf:** MDF handle.
- **pOffset:** Filter offset error compensation. This value is between Min_Data = -33554432 and Max_Data = 33554431.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_PollForSndLvl

Function name

HAL_StatusTypeDef HAL_MDF_PollForSndLvl (MDF_HandleTypeDef * hmdf, uint32_t Timeout, uint32_t * pSoundLevel, uint32_t * pAmbientNoise)

Function description

This function allows to poll for sound level data.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.
- **pSoundLevel:** Sound level. This parameter can be a value between Min_Data = 0 and Max_Data = 32767.
- **pAmbientNoise:** Ambient noise. This parameter can be a value between Min_Data = 0 and Max_Data = 32767.

Return values

- **HAL:** status.

Notes

- This function must not be used with MDF instance.

HAL_MDF_PollForSad

Function name

HAL_StatusTypeDef HAL_MDF_PollForSad (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

This function allows to poll for sound activity detection.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status.

Notes

- This function must not be used with MDF instance.

HAL_MDF_AcqCpltCallback

Function name

void HAL_MDF_AcqCpltCallback (MDF_HandleTypeDef * hmdf)

Function description

MDF acquisition complete callback.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_AcqHalfCpltCallback

Function name

void HAL_MDF_AcqHalfCpltCallback (MDF_HandleTypeDef * hmdf)

Function description

MDF acquisition half complete callback.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_SndLvlCallback

Function name

void HAL_MDF_SndLvlCallback (MDF_HandleTypeDef * hmdf, uint32_t SoundLevel, uint32_t AmbientNoise)

Function description

MDF sound level callback.

Parameters

- **hmdf:** MDF handle.
- **SoundLevel:** Sound level value computed by sound activity detector. This parameter can be a value between Min_Data = 0 and Max_Data = 32767.
- **AmbientNoise:** Ambient noise value computed by sound activity detector. This parameter can be a value between Min_Data = 0 and Max_Data = 32767.

Return values

- **None.:**

HAL_MDF_SadCallback

Function name

void HAL_MDF_SadCallback (MDF_HandleTypeDef * hmdf)

Function description

MDF sound activity detector callback.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_PollForCkab

Function name

HAL_StatusTypeDef HAL_MDF_PollForCkab (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

This function allows to poll for the clock absence detection.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status.

HAL_MDF_CkabStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_CkabStart_IT (MDF_HandleTypeDef * hmdf)

Function description

This function allows to start clock absence detection in interrupt mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_CkabStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_CkabStop_IT (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop clock absence detection in interrupt mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

HAL_MDF_ScdStart

Function name

HAL_StatusTypeDef HAL_MDF_ScdStart (MDF_HandleTypeDef * hmdf, const MDF_ScdConfigTypeDef * pScdConfig)

Function description

This function allows to start short-circuit detection in polling mode.

Parameters

- **hmdf:** MDF handle.
- **pScdConfig:** Short-circuit detector configuration parameters.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_PollForScd

Function name

HAL_StatusTypeDef HAL_MDF_PollForScd (MDF_HandleTypeDef * hmdf, uint32_t Timeout)

Function description

This function allows to poll for the short-circuit detection.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_ScdStop

Function name

HAL_StatusTypeDef HAL_MDF_ScdStop (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop short-circuit detection in polling mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_ScdStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_ScdStart_IT (MDF_HandleTypeDef * hmdf, const MDF_ScdConfigTypeDef * pScdConfig)

Function description

This function allows to start short-circuit detection in interrupt mode.

Parameters

- **hmdf:** MDF handle.
- **pScdConfig:** Short-circuit detector configuration parameters.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_ScdStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_ScdStop_IT (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop short-circuit detection in interrupt mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_OldStart

Function name

HAL_StatusTypeDef HAL_MDF_OldStart (MDF_HandleTypeDef * hmdf, const MDF_OldConfigTypeDef * pOldConfig)

Function description

This function allows to start out-off limit detection in polling mode.

Parameters

- **hmdf:** MDF handle.
- **pOldConfig:** Out-off limit detector configuration parameters.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_PollForOld

Function name

HAL_StatusTypeDef HAL_MDF_PollForOld (MDF_HandleTypeDef * hmdf, uint32_t Timeout, uint32_t * pThresholdInfo)

Function description

This function allows to poll for the out-off limit detection.

Parameters

- **hmdf:** MDF handle.
- **Timeout:** Timeout value in milliseconds.
- **pThresholdInfo:** Threshold information of out-off limit detection. This parameter can be a value of MDF out-off limit detector threshold information.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_OldStop

Function name

HAL_StatusTypeDef HAL_MDF_OldStop (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop out-off limit detection in polling mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_OldStart_IT

Function name

HAL_StatusTypeDef HAL_MDF_OldStart_IT (MDF_HandleTypeDef * hmdf, const MDF_OldConfigTypeDef * pOldConfig)

Function description

This function allows to start out-off limit detection in interrupt mode.

Parameters

- **hmdf:** MDF handle.
- **pOldConfig:** Out-off limit detector configuration parameters.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_OldStop_IT

Function name

HAL_StatusTypeDef HAL_MDF_OldStop_IT (MDF_HandleTypeDef * hmdf)

Function description

This function allows to stop out-off limit detection in interrupt mode.

Parameters

- **hmdf:** MDF handle.

Return values

- **HAL:** status.

Notes

- This function must not be used with ADF instance.

HAL_MDF_OldCallback

Function name

void HAL_MDF_OldCallback (MDF_HandleTypeDef * hmdf, uint32_t ThresholdInfo)

Function description

MDF out-off limit detector callback.

Parameters

- **hmdf:** MDF handle.
- **ThresholdInfo:** Threshold information of out-off limit detection. This parameter can be a value of MDF out-off limit detector threshold information.

Return values

- **None.:**

HAL_MDF_IRQHandler

Function name

void HAL_MDF_IRQHandler (MDF_HandleTypeDef * hmdf)

Function description

This function handles the MDF interrupts.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_ErrorCallback

Function name

void HAL_MDF_ErrorCallback (MDF_HandleTypeDef * hmdf)

Function description

MDF error callback.

Parameters

- **hmdf:** MDF handle.

Return values

- **None.:**

HAL_MDF_GetState

Function name

HAL_MDF_StateTypeDef HAL_MDF_GetState (const MDF_HandleTypeDef * hmdf)

Function description

This function allows to get the current MDF state.

Parameters

- **hmdf:** MDF handle.

Return values

- **MDF:** state.

HAL_MDF_GetError

Function name

uint32_t HAL_MDF_GetError (const MDF_HandleTypeDef * hmdf)

Function description

This function allows to get the current MDF error.

Parameters

- **hmdf:** MDF handle.

Return values

- **MDF:** error code.

48.3 MDF Firmware driver defines

The following section lists the various define and macros of the module.

48.3.1 MDF

MDF

MDF acquisition mode

MDF_MODE_ASYNC_CONT

Asynchronous, continuous acquisition mode

MDF_MODE_ASYNC_SINGLE

Notes:

- Not available for ADF instance with SAD usage

Asynchronous, single-shot acquisition mode.

MDF_MODE_SYNC_CONT

Synchronous, continuous acquisition mode

MDF_MODE_SYNC_SINGLE

Notes:

- Not available for ADF instance with SAD usage

Synchronous, single-shot acquisition mode.

MDF_MODE_WINDOW_CONT

Notes:

- Not available for ADF instance with SAD usage

Window, continuous acquisition mode.

MDF_MODE_SYNC_SNAPSHOT

Notes:

- Not available for ADF instance

Synchronous, snapshot acquisition mode.

MDF break signals

MDF_NO_BREAK_SIGNAL

No break signal

MDF_TIM1_BREAK_SIGNAL

TIM1_BREAK signal

MDF_TIM1_BREAK2_SIGNAL

TIM1_BREAK2 signal

MDF_TIM8_BREAK_SIGNAL

TIM8_BREAK signal

MDF_TIM8_BREAK2_SIGNAL

TIM8_BREAK2 signal

MDF CIC mode

MDF_TWO_FILTERS_MCIC_FASTSINC

Notes:

- Not available for ADF instance

Two filters, main filter in FastSinc order.

MDF_TWO_FILTERS_MCIC_SINC1

Notes:

- Not available for ADF instance

Two filters, main filter in Sinc1 order.

MDF_TWO_FILTERS_MCIC_SINC2

Notes:

- Not available for ADF instance

Two filters, main filter in Sinc2 order.

MDF_TWO_FILTERS_MCIC_SINC3

Notes:

- Not available for ADF instance

Two filters, main filter in Sinc3 order.

MDF_ONE_FILTER_SINC4

One filter in Sinc4 order

MDF_ONE_FILTER_SINC5

One filter in Sinc5 order

MDF output clock trigger edge

MDF_CLOCK_TRIG_RISING_EDGE

Rising edge

MDF_CLOCK_TRIG_FALLING_EDGE

Falling edge

MDF output clock trigger source

MDF_CLOCK_TRIG_TRGO

MDF_CLOCK_TRIG_TIM1_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM1_TRGO2

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM8_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM8_TRGO2

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM3_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM4_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM16_OC1

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM6_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_TIM7_TRGO

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_SAD_DET

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_EXTI11

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_EXTI15

MDF_CLOCK_TRIG_LPTIM1_OUT

Notes:

- Not available for ADF instance

MDF_CLOCK_TRIG_SAD_TRGO

Notes:

- Not available for ADF instance

MDF data source

MDF_DATA_SOURCE_BSMX

Data from bitstream matrix

MDF_DATA_SOURCE_ADCITF1

Notes:

- Not available for ADF instance

Data from ADC interface 1.

MDF_DATA_SOURCE_ADCITF2

Notes:

- Not available for ADF instance

Data from ADC interface 2.

MDF error code

MDF_ERROR_NONE

No error

MDF_ERROR_ACQUISITION_OVERFLOW

Overflow occurs during acquisition

MDF_ERROR_RSF_OVERRUN

Overrun occurs on reshape filter

MDF_ERROR_CLOCK_ABSENCE

Clock absence detection occurs

MDF_ERROR_SHORT_CIRCUIT

Notes:

- Not used for ADF instance

Short circuit detection occurs.

MDF_ERROR_SATURATION

Saturation detection occurs

MDF_ERROR_OUT_OFF_LIMIT

Notes:

- Not used for ADF instance

Out-off limit detection occurs.

MDF_ERROR_DMA

DMA error occurs

MDF_ERROR_INVALID_CALLBACK

Invalid callback error occurs

MDF Exported Macros

__HAL_MDF_RESET_HANDLE_STATE

Description:

- Reset MDF handle state.

Parameters:

- __HANDLE__: MDF handle.

Return value:

- None

MDF RXFIFO threshold

MDF_FIFO_THRESHOLD_NOT_EMPTY

Event generated when RXFIFO is not empty

MDF_FIFO_THRESHOLD_HALF_FULL

Event generated when RXFIFO is half_full

MDF filter bitstream

MDF_BITSTREAM0_RISING

MDF_BITSTREAM0_FALLING

MDF_BITSTREAM1_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM1_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM2_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM2_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM3_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM3_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM4_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM4_FALLING

Notes:

- Not available for ADF instance

MDF_BITSTREAM5_RISING

Notes:

- Not available for ADF instance

MDF_BITSTREAM5_FALLING

Notes:

- Not available for ADF instance

MDF filter trigger edge

MDF_FILTER_TRIG_RISING_EDGE

Rising edge

MDF_FILTER_TRIG_FALLING_EDGE

Falling edge

MDF filter trigger source

MDF_FILTER_TRIG_TRGO

MDF_FILTER_TRIG_OLD_EVENT

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM1_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM1_TRGO2

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM8_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM8_TRGO2

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM3_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM4_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM16_OC1

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM6_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_TIM7_TRGO

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_ADF_SAD_DET

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_EXTI11

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_EXTI15

MDF_FILTER_TRIG_LPTIM1_OUT

Notes:

- Not available for ADF instance

MDF_FILTER_TRIG_ADF_TRGO

Notes:

- Not available for ADF instance

MDF high pass filter cut-off frequency

MDF_HPF_CUTOFF_0_000625FPCM

Cut-off frequency of 0.000625xFpcm

MDF_HPF_CUTOFF_0_00125FPCM

Cut-off frequency of 0.00125xFpcm

MDF_HPF_CUTOFF_0_0025FPCM

Cut-off frequency of 0.0025xFpcm

MDF_HPF_CUTOFF_0_0095FPCM

Cut-off frequency of 0.0095xFpcm

MDF integrator output division

MDF_INTEGRATOR_OUTPUT_DIV_128

Integrator data outputs divided by 128

MDF_INTEGRATOR_OUTPUT_DIV_32

Integrator data outputs divided by 32

MDF_INTEGRATOR_OUTPUT_DIV_4

Integrator data outputs divided by 4

MDF_INTEGRATOR_OUTPUT_NO_DIV

Integrator data outputs not divided

MDF out-off limit detector CIC mode

MDF_OLD_FILTER_FASTSINC

Out-off limit detector filter in FastSinc order

MDF_OLD_FILTER_SINC1

Out-off limit detector filter in Sinc1 order

MDF_OLD_FILTER_SINC2

Out-off limit detector filter in Sinc2 order

MDF_OLD_FILTER_SINC3

Out-off limit detector filter in Sinc3 order

MDF out-off limit detector event configuration

MDF_OLD_SIGNAL_OUTSIDE_THRESHOLDS

Out-off limit detector event is generated if signal is outside thresholds

MDF_OLD_SIGNAL_IN_THRESHOLDS

Out-off limit detector event is generated if signal is in thresholds

MDF out-off limit detector threshold information

MDF_OLD_IN_THRESHOLDS

The signal was in thresholds when out-off limit detection occurs

MDF_OLD_HIGH_THRESHOLD

The signal was higher than high threshold when out-off limit detection occurs

MDF_OLD_LOW_THRESHOLD

The signal was lower than low threshold when out-off limit detection occurs

MDF output clock pins

MDF_OUTPUT_CLOCK_0

MDF_CCK0 is used as output clock

MDF_OUTPUT_CLOCK_1

MDF_CCK1 is used as output clock

MDF_OUTPUT_CLOCK_ALL

MDF_CCK0 and MDF_CCK1 are used as output clock

MDF reshape filter decimation ratio

MDF_RSF_DECIMATION_RATIO_4

Reshape filter decimation ratio is 4

MDF_RSF_DECIMATION_RATIO_1

Reshape filter decimation ratio is 1

MDF sound activity detector data memory transfer mode

MDF_SAD_NO_MEMORY_TRANSFER

No memory transfer

MDF_SAD_MEMORY_TRANSFER_IN_DETECT

Memory transfer only in detect state

MDF_SAD_MEMORY_TRANSFER_ALWAYS

Memory transfer always

MDF sound activity detector frame size

MDF_SAD_8_PCM_SAMPLES

Frame size of 8 PCM samples

MDF_SAD_16_PCM_SAMPLES

Frame size of 16 PCM samples

MDF_SAD_32_PCM_SAMPLES

Frame size of 32 PCM samples

MDF_SAD_64_PCM_SAMPLES

Frame size of 64 PCM samples

MDF_SAD_128_PCM_SAMPLES

Frame size of 128 PCM samples

MDF_SAD_256_PCM_SAMPLES

Frame size of 256 PCM samples

MDF_SAD_512_PCM_SAMPLES

Frame size of 512 PCM samples

MDF sound activity detector data hangover time window

MDF_SAD_HANGOVER_4_FRAMES

Hangover window of 4 frames

MDF_SAD_HANGOVER_8_FRAMES

Hangover window of 8 frames

MDF_SAD_HANGOVER_16_FRAMES

Hangover window of 16 frames

MDF_SAD_HANGOVER_32_FRAMES

Hangover window of 32 frames

MDF_SAD_HANGOVER_64_FRAMES

Hangover window of 64 frames

MDF_SAD_HANGOVER_128_FRAMES

Hangover window of 128 frames

MDF_SAD_HANGOVER_256_FRAMES

Hangover window of 256 frames

MDF_SAD_HANGOVER_512_FRAMES

Hangover window of 512 frames

MDF sound activity detector data learning frames

MDF_SAD_LEARNING_2_FRAMES

2 learning frames

MDF_SAD_LEARNING_4_FRAMES

4 learning frames

MDF_SAD_LEARNING_8_FRAMES

8 learning frames

MDF_SAD_LEARNING_16_FRAMES

16 learning frames

MDF_SAD_LEARNING_32_FRAMES
32 learning frames

MDF sound activity detector mode

MDF_SAD_VOICE_ACTIVITY_DETECTOR
Voice activity detector

MDF_SAD_SOUND_DETECTOR
Sound detector

MDF_SAD_AMBIENT_NOISE_DETECTOR
Ambient noise detector

MDF sound activity detector data signal to noise threshold

MDF_SAD_SIGNAL_NOISE_3_5DB
Signal to noise threshold is 3.5dB

MDF_SAD_SIGNAL_NOISE_6DB
Signal to noise threshold is 6dB

MDF_SAD_SIGNAL_NOISE_9_5DB
Signal to noise threshold is 9.5dB

MDF_SAD_SIGNAL_NOISE_12DB
Signal to noise threshold is 12dB

MDF_SAD_SIGNAL_NOISE_15_6DB
Signal to noise threshold is 15.6dB

MDF_SAD_SIGNAL_NOISE_18DB
Signal to noise threshold is 18dB

MDF_SAD_SIGNAL_NOISE_21_6DB
Signal to noise threshold is 21.6dB

MDF_SAD_SIGNAL_NOISE_24_1DB
Signal to noise threshold is 24.1dB

MDF_SAD_SIGNAL_NOISE_27_6DB
Signal to noise threshold is 27.6dB

MDF_SAD_SIGNAL_NOISE_30_1DB
Signal to noise threshold is 30.1dB

MDF sound activity detector trigger event

MDF_SAD_ENTER_DETECT
Event when SAD enters in detect state

MDF_SAD_ENTER_EXIT_DETECT
Event when SAD enters or exits from detect state

MDF serial interface clock source

MDF_SITF_CCK0_SOURCE
Common clock 0 source

MDF_SITF_CCK1_SOURCE

Common clock 1 source

MDF_SITF_CKI_SOURCE

Notes:

- Not available for ADF instance

Dedicated input clock source.

MDF serial interface mode

MDF_SITF_LF_MASTER_SPI_MODE

Low frequency master SPI mode

MDF_SITF_NORMAL_SPI_MODE

Normal SPI mode

MDF_SITF_MANCHESTER_FALLING_MODE

Manchester mode rising edge logic 0 and falling edge logic 1

MDF_SITF_MANCHESTER_RISING_MODE

Manchester mode rising edge logic 1 and falling edge logic 0

MDF snapshot format

MDF_SNAPSHOT_23BITS

Snapshot data resolution of 23 bits

MDF_SNAPSHOT_16BITS

Snapshot data resolution of 16 bits

49 HAL MMC Generic Driver

49.1 MMC Firmware driver registers structures

49.1.1 HAL_MMC_CardInfoTypeDef

HAL_MMC_CardInfoTypeDef is defined in the stm32u5xx_hal_mmc.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***

Field Documentation

- ***uint32_t HAL_MMC_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_MMC_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_MMC_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_MMC_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes

49.1.2 __MMC_HandleTypeDef

__MMC_HandleTypeDef is defined in the stm32u5xx_hal_mmc.h

Data Fields

- ***MMC_TypeDef * Instance***
- ***MMC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_MMC_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***__IO uint16_t RPMBErrorCode***
- ***HAL_MMC_CardInfoTypeDef MmcCard***
- ***uint32_t CSD***
- ***uint32_t CID***
- ***uint32_t Ext_CSD***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***

- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* Read_DMALnkLstBufCpltCallback*
- *void(* Write_DMALnkLstBufCpltCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***MMC_TypeDef* __MMC_HandleTypeDef::Instance***
  MMC registers base address
- ***MMC_InitTypeDef __MMC_HandleTypeDef::Init***
  MMC required parameters
- ***HAL_LockTypeDef __MMC_HandleTypeDef::Lock***
  MMC locking object
- ***const uint8_t* __MMC_HandleTypeDef::pTxBuffPtr***
  Pointer to MMC Tx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::TxXferSize***
  MMC Tx Transfer size
- ***uint8_t* __MMC_HandleTypeDef::pRxBuffPtr***
  Pointer to MMC Rx transfer Buffer
- ***uint32_t __MMC_HandleTypeDef::RxXferSize***
  MMC Rx Transfer size
- ***__IO uint32_t __MMC_HandleTypeDef::Context***
  MMC transfer context
- ***__IO HAL_MMC_StateTypeDef __MMC_HandleTypeDef::State***
  MMC card State
- ***__IO uint32_t __MMC_HandleTypeDef::ErrorCode***
  MMC Card Error codes
- ***__IO uint16_t __MMC_HandleTypeDef::RPMBErrorCode***
  MMC RPMB Area Error codes
- ***HAL_MMC_CardInfoTypeDef __MMC_HandleTypeDef::MmcCard***
  MMC Card information
- ***uint32_t __MMC_HandleTypeDef::CSD[4U]***
  MMC card specific data table
- ***uint32_t __MMC_HandleTypeDef::CID[4U]***
  MMC card identification number table
- ***uint32_t __MMC_HandleTypeDef::Ext_CSD[128]***
- ***void(* __MMC_HandleTypeDef::TxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::RxCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::ErrorCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::AbortCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Read_DMALnkLstBufCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::Write_DMALnkLstBufCpltCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::MspInitCallback)(struct __MMC_HandleTypeDef *hmmc)***
- ***void(* __MMC_HandleTypeDef::MspDeInitCallback)(struct __MMC_HandleTypeDef *hmmc)***

49.1.3 HAL_MMC_CardCSDTypeDef

HAL_MMC_CardCSDTypeDef is defined in the stm32u5xx_hal_mmc.h

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***

- ***__IO uint8_t TAAC***
- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGroup***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_MMC_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockLen***
  Max. read data block length

- **__IO uint8_t HAL_MMC_CardCSDTypeDef::PartBlockRead**
  Partial blocks for read allowed
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::WrBlockMisalign**
  Write block misalignment
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::RdBlockMisalign**
  Read block misalignment
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::DSRImpl**
  DSR implemented
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved2**
  Reserved
- **__IO uint32_t HAL_MMC_CardCSDTypeDef::DeviceSize**
  Device Size
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMin**
  Max. read current @ VDD min
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxRdCurrentVDDMax**
  Max. read current @ VDD max
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMin**
  Max. write current @ VDD min
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrCurrentVDDMax**
  Max. write current @ VDD max
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::DeviceSizeMul**
  Device size multiplier
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrSize**
  Erase group size
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::EraseGrMul**
  Erase group size multiplier
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrSize**
  Write protect group size
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::WrProtectGrEnable**
  Write protect group enable
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::ManDeflECC**
  Manufacturer default ECC
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::WrSpeedFact**
  Write speed factor
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::MaxWrBlockLen**
  Max. write data block length
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::WriteBlockPaPartial**
  Partial blocks for write allowed
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved3**
  Reserved
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::ContentProtectAppli**
  Content protection application
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormatGroup**
  File format group
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::CopyFlag**
  Copy flag (OTP)
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::PermWrProtect**
  Permanent write protection
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::TempWrProtect**
  Temporary write protection
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::FileFormat**
  File format
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::ECC**
  ECC code
- **__IO uint8_t HAL_MMC_CardCSDTypeDef::CSD_CRC**
  CSD CRC

- ***__IO uint8_t HAL_MMC_CardCSDTypeDef::Reserved4***
  Always 1

49.1.4 HAL_MMC_CardCIDTypeDef

HAL_MMC_CardCIDTypeDef is defined in the stm32u5xx_hal_mmc.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_MMC_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_MMC_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_MMC_CardCIDTypeDef::Reserved2***
  Always 1

49.2 MMC Firmware driver API description

The following section lists the various functions of the MMC library.

49.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_MMC_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with MMC and eMMC cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implement the HAL_MMC_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for MMC card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_MMC_ReadBlocks_IT() and HAL_MMC_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_MMC_ENABLE_IT() and __HAL_MMC_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_MMC_GET_IT() and __HAL_MMC_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform MMC read/write/erase operations after MMC card initialization

MMC Card Initialization and configuration

To initialize the MMC Card, use the HAL_MMC_Init() function. It Initializes SDMMC Peripheral (STM32 side) and the MMC Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Initialize the SDMMC peripheral interface with default configuration. The initialization process is done at 400KHz. You can change or adapt this frequency by adjusting the "ClockDiv" field. The MMC Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the MMC card. The API used is HAL_MMC_InitCard(). This phase allows the card initialization and identification and check the MMC Card type (Standard Capacity or High Capacity) The initialization flow is compatible with MMC standard. This API (HAL_MMC_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the MMC Card Data transfer frequency. By Default, the card transfer frequency by adjusting the "ClockDiv" field. In transfer mode and according to the MMC Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding MMC Card according to the address read with the step 2.
5. Configure the MMC Card in wide bus mode: 4-bits data.
6. Select the MMC Card partition using HAL_MMC_SwitchPartition()

MMC Card Read operation

- You can read from MMC card in polling mode by using function HAL_MMC_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can read from MMC card in DMA mode by using function HAL_MMC_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Rx interrupt event.
- You can read from MMC card in Interrupt mode by using function HAL_MMC_ReadBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Rx interrupt event.

MMC Card Write operation

- You can write to MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state.
- You can write to MMC card in DMA mode by using function HAL_MMC_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the DMA transfer process through the MMC Tx interrupt event.
- You can write to MMC card in Interrupt mode by using function HAL_MMC_WriteBlocks_IT(). This function allows the read of 512 bytes blocks. You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetCardState() function for MMC card state. You could also check the IT transfer process through the MMC Tx interrupt event.

MMC card information

- To get MMC card information, you can use the function HAL_MMC_GetCardInfo(). It returns useful information about the MMC card such as block size, card type, block number ...

MMC card CSD register

- The HAL_MMC_GetCardCSD() API allows to get the parameters of the CSD register. Some of the CSD parameters are useful for card initialization and identification.

MMC card CID register

- The HAL_MMC_GetCardCID() API allows to get the parameters of the CID register. Some of the CID parameters are useful for card initialization and identification.

MMC Card Reply Protected Memory Block (RPMB) Key Programming operation

- You can program the authentication key of RPMB area in polling mode by using function HAL_MMC_RPMB_ProgramAuthenticationKey(). This function is only used once during the life of an MMC card. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.
- You can program the authentication key of RPMB area in Interrupt mode by using function HAL_MMC_RPMB_ProgramAuthenticationKey_IT(). This function is only used once during the life of an MMC card. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.

MMC Card Reply Protected Memory Block (RPMB) write counter operation

- You can get the write counter value of RPMB area in polling mode by using function HAL_MMC_RPMB_GetWriteCounter().
- You can get the write counter value of RPMB area in Interrupt mode by using function HAL_MMC_RPMB_GetWriteCounter_IT().

MMC Card Reply Protected Memory Block (RPMB) write operation

- You can write to the RPMB area of MMC card in polling mode by using function HAL_MMC_WriteBlocks(). This function supports the one, two, or thirty two blocks write operation (with 512-bytes block length). You can choose the number of blocks at the multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.

- You can write to the RPMB area of MMC card in Interrupt mode by using function HAL_MMC_WriteBlocks_IT(). This function supports the one, two, or thirty two blocks write operation (with 512-bytes block length). You can choose the number of blocks at the multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for operation state.

MMC Card Reply Protected Memory Block (RPMB) read operation

- You can read from the RPMB area of MMC card in polling mode by using function HAL_MMC_RPMB_ReadBlocks(). The block size should be chosen as multiple of 512 bytes. You can choose the number of blocks by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for MMC card state.
- You can read from the RPMB area of MMC card in Interrupt mode by using function HAL_MMC_RPMB_ReadBlocks_IT(). The block size should be chosen as multiple of 512 bytes. You can choose the number of blocks by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_MMC_GetRPMBError() function for MMC card state.

MMC HAL driver macros list

Below the list of most used macros in MMC HAL driver.

- __HAL_MMC_ENABLE_IT: Enable the MMC device interrupt
- __HAL_MMC_DISABLE_IT: Disable the MMC device interrupt
- __HAL_MMC_GET_FLAG:Check whether the specified MMC flag is set or not
- __HAL_MMC_CLEAR_FLAG: Clear the MMC's pending flags

Note: You can refer to the MMC HAL driver header file for more useful macros

Callback registration

The compilation define USE_HAL_MMC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_MMC_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMALnkLstBufCpltCallback : callback when the DMA reception of linked list node buffer is completed.
- Write_DMALnkLstBufCpltCallback : callback when the DMA transmission of linked list node buffer is completed.
- MspInitCallback : MMC MspInit.
- MspDeInitCallback : MMC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_MMC_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : MMC MspInit.

- MspDeInitCallback : MMC MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_MMC_Init and if the state is HAL_MMC_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_MMC_Init and HAL_MMC_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_MMC_Init and HAL_MMC_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_MMC_RegisterCallback before calling HAL_MMC_DeInit or HAL_MMC_Init function. When The compilation define USE_HAL_MMC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

49.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the MMC card device to be ready for use.

This section contains the following APIs:

- *HAL_MMC_Init()*
- *HAL_MMC_InitCard()*
- *HAL_MMC_DeInit()*
- *HAL_MMC_MspInit()*
- *HAL_MMC_MspDeInit()*

49.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to MMC card.

This section contains the following APIs:

- *HAL_MMC_ReadBlocks()*
- *HAL_MMC_WriteBlocks()*
- *HAL_MMC_ReadBlocks_IT()*
- *HAL_MMC_WriteBlocks_IT()*
- *HAL_MMC_ReadBlocks_DMA()*
- *HAL_MMC_WriteBlocks_DMA()*
- *HAL_MMC_Erase()*
- *HAL_MMC_IRQHandler()*
- *HAL_MMC_GetState()*
- *HAL_MMC_GetError()*
- *HAL_MMC_TxCpltCallback()*
- *HAL_MMC_RxCpltCallback()*
- *HAL_MMC_ErrorCallback()*
- *HAL_MMC_AbortCallback()*
- *HAL_MMC_RegisterCallback()*
- *HAL_MMC_UnRegisterCallback()*

49.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the MMC card operations and get the related information

This section contains the following APIs:

- *HAL_MMC_GetCardCID()*
- *HAL_MMC_GetCardCSD()*
- *HAL_MMC_GetCardInfo()*
- *HAL_MMC_GetCardExtCSD()*
- *HAL_MMC_ConfigWideBusOperation()*

- *HAL_MMC_ConfigSpeedBusOperation()*
- *HAL_MMC_GetCardState()*
- *HAL_MMC_Abort()*
- *HAL_MMC_Abort_IT()*
- *HAL_MMC_EraseSequence()*
- *HAL_MMC_Sanitize()*
- *HAL_MMC_ConfigSecRemovalType()*
- *HAL_MMC_GetSupportedSecRemovalType()*
- *HAL_MMC_SleepDevice()*
- *HAL_MMC_AwakeDevice()*
- *HAL_MMC_SwitchPartition()*

49.2.5 Detailed description of functions

HAL_MMC_Init

Function name

HAL_StatusTypeDef HAL_MMC_Init (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC according to the specified parameters in the MMC_HandleTypeDef and create the associated handle.

Parameters

- **hmmc:** Pointer to the MMC handle

Return values

- **HAL:** status

HAL_MMC_InitCard

Function name

HAL_StatusTypeDef HAL_MMC_InitCard (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC Card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This function initializes the MMC card. It could be used when a card re-initialization is needed.

HAL_MMC_DeInit

Function name

HAL_StatusTypeDef HAL_MMC_DeInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initializes the MMC card.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_MspInit

Function name

void HAL_MMC_MspInit (MMC_HandleTypeDef * hmmc)

Function description

Initializes the MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_MspDeInit

Function name

void HAL_MMC_MspDeInit (MMC_HandleTypeDef * hmmc)

Function description

De-Initialize MMC MSP.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of MMC blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of MMC blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Erase

Function name

HAL_StatusTypeDef HAL_MMC_Erase (MMC_HandleTypeDef * hmmc, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given MMC card.

Parameters

- **hmmc:** Pointer to MMC handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the IT transfer process through the MMC Tx interrupt event.

HAL_MMC_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_ReadBlocks_DMA (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hmmc:** Pointer MMC handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Rx interrupt event.

HAL_MMC_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_MMC_WriteBlocks_DMA (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().
- You could also check the DMA transfer process through the MMC Tx interrupt event.

HAL_MMC_IRQHandler

Function name

void HAL_MMC_IRQHandler (MMC_HandleTypeDef * hmmc)

Function description

This function handles MMC card interrupt request.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_TxCpltCallback

Function name

void HAL_MMC_TxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Tx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **None:**

HAL_MMC_RxCpltCallback

Function name

void HAL_MMC_RxCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Rx Transfer completed callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_ErrorCallback

Function name

void HAL_MMC_ErrorCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC error callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_AbortCallback

Function name

void HAL_MMC_AbortCallback (MMC_HandleTypeDef * hmmc)

Function description

MMC Abort callbacks.

Parameters

- **hmmc:** Pointer MMC handle

Return values

- **None:**

HAL_MMC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_RegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId, pMMC_CallbackTypeDef pCallback)

Function description

Register a User MMC Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hmmc:** : MMC handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_MMC_TX_CPLT_CB_ID MMC Tx Complete Callback ID
  - HAL_MMC_RX_CPLT_CB_ID MMC Rx Complete Callback ID
  - HAL_MMC_ERROR_CB_ID MMC Error Callback ID
  - HAL_MMC_ABORT_CB_ID MMC Abort Callback ID
  - HAL_MMC_READ_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Rx Linked List Node buffer Callback ID
  - HAL_MMC_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Tx Linked List Node buffer Callback ID
  - HAL_MMC_MSP_INIT_CB_ID MMC MspInit Callback ID
  - HAL_MMC_MSP_DEINIT_CB_ID MMC MspDeInit Callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_MMC_RegisterCallback() may be called before HAL_MMC_Init() in HAL_MMC_STATE_RESET to register callbacks for HAL_MMC_MSP_INIT_CB_ID and HAL_MMC_MSP_DEINIT_CB_ID.

HAL_MMC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_MMC_UnRegisterCallback (MMC_HandleTypeDef * hmmc, HAL_MMC_CallbackIDTypeDef CallbackId)

Function description

Unregister a User MMC Callback MMC Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hmmc:** : MMC handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_MMC_TX_CPLT_CB_ID MMC Tx Complete Callback ID
  - HAL_MMC_RX_CPLT_CB_ID MMC Rx Complete Callback ID
  - HAL_MMC_ERROR_CB_ID MMC Error Callback ID
  - HAL_MMC_ABORT_CB_ID MMC Abort Callback ID
  - HAL_MMC_READ_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Rx Linked List Node buffer Callback ID
  - HAL_MMC_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID MMC DMA Tx Linked List Node buffer Callback ID
  - HAL_MMC_MSP_INIT_CB_ID MMC MspInit Callback ID
  - HAL_MMC_MSP_DEINIT_CB_ID MMC MspDeInit Callback ID

Return values

- **status:**

Notes

- The HAL_MMC_UnRegisterCallback() may be called before HAL_MMC_Init() in HAL_MMC_STATE_RESET to register callbacks for HAL_MMC_MSP_INIT_CB_ID and HAL_MMC_MSP_DEINIT_CB_ID.

HAL_MMC_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigWideBusOperation (MMC_HandleTypeDef * hmmc, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hmmc:** Pointer to MMC handle
- **WideMode:** Specifies the MMC card wide bus mode This parameter can be one of the following values:
  - SDMMC_BUS_WIDE_8B: 8-bit data transfer
  - SDMMC_BUS_WIDE_4B: 4-bit data transfer
  - SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_MMC_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSpeedBusOperation (MMC_HandleTypeDef * hmmc, uint32_t SpeedMode)

Function description

Configure the speed bus mode.

Parameters

- **hmmc:** Pointer to the MMC handle
- **SpeedMode:** Specifies the MMC card speed bus mode This parameter can be one of the following values:
  - SDMMC_SPEED_MODE_AUTO: Max speed mode supported by the card
  - SDMMC_SPEED_MODE_DEFAULT: Default Speed (MMC @ 26MHz)
  - SDMMC_SPEED_MODE_HIGH: High Speed (MMC @ 52 MHz)
  - SDMMC_SPEED_MODE_DDR: High Speed DDR (MMC DDR @ 52 MHz)

Return values

- **HAL:** status

HAL_MMC_SwitchPartition

Function name

HAL_StatusTypeDef HAL_MMC_SwitchPartition (MMC_HandleTypeDef * hmmc, HAL_MMC_PartitionTypeDef Partition)

Function description

Used to select the partition.

Parameters

- **hmmc:** Pointer to MMC handle
- **Partition:** Partition type
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_GetCardState

Function name

HAL_MMC_CardStateTypeDef HAL_MMC_GetCardState (MMC_HandleTypeDef * hmmc)

Function description

Gets the current mmc card data state.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **Card:** state

HAL_MMC_GetCardCID

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCID (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCID:** Pointer to a HAL_MMC_CIDTypedef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardCSD (MMC_HandleTypeDef * hmmc, HAL_MMC_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCSD:** Pointer to a HAL_MMC_CardCSDTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_MMC_GetCardInfo

Function name

HAL_StatusTypeDef HAL_MMC_GetCardInfo (MMC_HandleTypeDef * hmmc, HAL_MMC_CardInfoTypeDef * pCardInfo)

Function description

Gets the MMC card info.

Parameters

- **hmmc:** Pointer to MMC handle
- **pCardInfo:** Pointer to the HAL_MMC_CardInfoTypeDef structure that will contain the MMC card status information

Return values

- **HAL:** status

HAL_MMC_GetCardExtCSD

Function name

HAL_StatusTypeDef HAL_MMC_GetCardExtCSD (MMC_HandleTypeDef * hmmc, uint32_t * pExtCSD, uint32_t Timeout)

Function description

Returns information the information of the card which are stored on the Extended CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **pExtCSD:** Pointer to a memory area (512 bytes) that contains all Extended CSD register parameters
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_GetState

Function name

HAL_MMC_StateTypeDef HAL_MMC_GetState (const MMC_HandleTypeDef * hmmc)

Function description

return the MMC state

Parameters

- **hmmc:** Pointer to mmc handle

Return values

- **HAL:** state

HAL_MMC_GetError

Function name

uint32_t HAL_MMC_GetError (const MMC_HandleTypeDef * hmmc)

Function description

Return the MMC error code.

Parameters

- **hmmc:** : Pointer to a MMC_HandleTypeDef structure that contains the configuration information.

Return values

- **MMC:** Error Code

HAL_MMC_GetRPMBError

Function name

uint32_t HAL_MMC_GetRPMBError (const MMC_HandleTypeDef * hmmc)

Function description

HAL_MMC_Abort

Function name

HAL_StatusTypeDef HAL_MMC_Abort (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC.

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_Abort_IT

Function name

HAL_StatusTypeDef HAL_MMC_Abort_IT (MMC_HandleTypeDef * hmmc)

Function description

Abort the current transfer and disable the MMC (IT mode).

Parameters

- **hmmc:** pointer to a MMC_HandleTypeDef structure that contains the configuration information for MMC module.

Return values

- **HAL:** status

HAL_MMC_EraseSequence

Function name

HAL_StatusTypeDef HAL_MMC_EraseSequence (MMC_HandleTypeDef * hmmc, uint32_t EraseType, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Perform specific commands sequence for the different type of erase.

Parameters

- **hmmc:** Pointer to MMC handle
- **EraseType:** Specifies the type of erase to be performed This parameter can be one of the following values:
  – HAL_MMC_TRIM Erase the write blocks identified by CMD35 & 36
  – HAL_MMC_ERASE Erase the erase groups identified by CMD35 & 36
  – HAL_MMC_DISCARD Discard the write blocks identified by CMD35 & 36
  – HAL_MMC_SECURE_ERASE Perform a secure purge according SRT on the erase groups identified by CMD35 & 36
  – HAL_MMC_SECURE_TRIM_STEP1 Mark the write blocks identified by CMD35 & 36 for secure erase
  – HAL_MMC_SECURE_TRIM_STEP2 Perform a secure purge according SRT on the write blocks previously identified
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_Sanitize

Function name

HAL_StatusTypeDef HAL_MMC_Sanitize (MMC_HandleTypeDef * hmmc)

Function description

Perform sanitize operation on the device.

Parameters

- **hmmc:** Pointer to MMC handle

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_ConfigSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_ConfigSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t SRTMode)

Function description

Configure the Secure Removal Type (SRT) in the Extended CSD register.

Parameters

- **hmmc:** Pointer to MMC handle
- **SRTMode:** Specifies the type of erase to be performed This parameter can be one of the following values:
  – HAL_MMC_SRT_ERASE Information removed by an erase
  – HAL_MMC_SRT_WRITE_CHAR_ERASE Information removed by an overwriting with a character followed by an erase
  – HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM Information removed by an overwriting with a character, its complement then a random character
  – HAL_MMC_SRT_VENDOR_DEFINED Information removed using a vendor defined

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_MMC_GetCardState().

HAL_MMC_GetSupportedSecRemovalType

Function name

HAL_StatusTypeDef HAL_MMC_GetSupportedSecRemovalType (MMC_HandleTypeDef * hmmc, uint32_t * SupportedSRT)

Function description

Gets the supported values of the the Secure Removal Type (SRT).

Parameters

- **hmmc:** pointer to MMC handle
- **SupportedSRT:** pointer for supported SRT value This parameter is a bit field of the following values:
  – HAL_MMC_SRT_ERASE Information removed by an erase
  – HAL_MMC_SRT_WRITE_CHAR_ERASE Information removed by an overwriting with a character followed by an erase
  – HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM Information removed by an overwriting with a character, its complement then a random character
  – HAL_MMC_SRT_VENDOR_DEFINED Information removed using a vendor defined

Return values

- **HAL:** status

HAL_MMC_SleepDevice

Function name

HAL_StatusTypeDef HAL_MMC_SleepDevice (MMC_HandleTypeDef * hmmc)

Function description

Switch the device from Standby State to Sleep State.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_AwakeDevice

Function name

HAL_StatusTypeDef HAL_MMC_AwakeDevice (MMC_HandleTypeDef * hmmc)

Function description

Switch the device from Sleep State to Standby State.

Parameters

- **hmmc:** pointer to MMC handle

Return values

- **HAL:** status

HAL_MMC_RPMB_ProgramAuthenticationKey

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ProgramAuthenticationKey (MMC_HandleTypeDef * hmmc, const uint8_t * pKey, uint32_t Timeout)

Function description

Allows to program the authentication key within the RPMB partition.

Parameters

- **hmmc:** Pointer to MMC handle
- **pKey:** pointer to the authentication key (32 bytes)
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_RPMB_ProgramAuthenticationKey_IT

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ProgramAuthenticationKey_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pKey, uint32_t Timeout)

Function description

HAL_MMC_RPMB_GetWriteCounter

Function name

uint32_t HAL_MMC_RPMB_GetWriteCounter (MMC_HandleTypeDef * hmmc, uint8_t * pNonce, uint32_t Timeout)

Function description

Allows to get the value of write counter within the RPMB partition.

Parameters

- **hmmc:** Pointer to MMC handle
- **Nonce:** pointer to the value of nonce (16 bytes)
- **Timeout:** Specify timeout value

Return values

- **write:** counter value.

HAL_MMC_RPMB_GetWriteCounter_IT

Function name

uint32_t HAL_MMC_RPMB_GetWriteCounter_IT (MMC_HandleTypeDef * hmmc, uint8_t * pNonce)

Function description

HAL_MMC_RPMB_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_WriteBlocks (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pMAC, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in the RPMB partition.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write
- **pMAC:** Pointer to the authentication MAC buffer
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_RPMB_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_WriteBlocks_IT (MMC_HandleTypeDef * hmmc, const uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pMAC)

Function description

HAL_MMC_RPMB_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ReadBlocks (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pNonce, uint8_t * pMAC, uint32_t Timeout)

Function description

Allows to read block(s) to a specified address in the RPMB partition.

Parameters

- **hmmc:** Pointer to MMC handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write
- **pNonce:** Pointer to the buffer that will contain the nonce to transmit
- **pMAC:** Pointer to the authentication MAC buffer
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

HAL_MMC_RPMB_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_MMC_RPMB_ReadBlocks_IT (MMC_HandleTypeDef * hmmc, uint8_t * pData, uint16_t BlockAdd, uint16_t NumberOfBlocks, const uint8_t * pNonce, uint8_t * pMAC)

Function description

49.3 MMC Firmware driver defines

The following section lists the various define and macros of the module.

49.3.1 MMC

MMC

MMC Error status enumeration Structure definition

HAL_MMC_ERROR_NONE

No error

HAL_MMC_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_MMC_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_MMC_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_MMC_ERROR_DATA_TIMEOUT

Data timeout

HAL_MMC_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_MMC_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_MMC_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_MMC_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_MMC_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_MMC_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_MMC_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_MMC_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_MMC_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_MMC_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_MMC_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_MMC_ERROR_CC_ERR

Internal card controller error

HAL_MMC_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_MMC_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_MMC_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_MMC_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_MMC_ERROR_WP_ERASE_SKIP

Only partial address space was erased

HAL_MMC_ERROR_CARD_ECC_DISABLED

Command has been executed without using internal ECC

HAL_MMC_ERROR_ERASE_RESET

Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_MMC_ERROR_AKE_SEQ_ERR

Error in sequence of authentication

HAL_MMC_ERROR_INVALID_VOLTRANGE
Error in case of invalid voltage range

HAL_MMC_ERROR_ADDR_OUT_OF_RANGE
Error when addressed block is out of range

HAL_MMC_ERROR_REQUEST_NOT_APPLICABLE
Error when command request is not applicable

HAL_MMC_ERROR_PARAM
the used parameter is not valid

HAL_MMC_ERROR_UNSUPPORTED_FEATURE
Error when feature is not insupported

HAL_MMC_ERROR_BUSY
Error when transfer process is busy

HAL_MMC_ERROR_DMA
Error while DMA transfer

HAL_MMC_ERROR_TIMEOUT
Timeout error response results after operating with RPMB partition

HAL_MMC_ERROR_RPMB_OPERATION_OK
Operation OK

HAL_MMC_ERROR_RPMB_GENERAL_FAILURE
General failure

HAL_MMC_ERROR_RPMB_AUTHENTICATION_FAILURE
Authentication failure

HAL_MMC_ERROR_RPMB_COUNTER_FAILURE
Counter failure

HAL_MMC_ERROR_RPMB_ADDRESS_FAILURE
Address failure

HAL_MMC_ERROR_RPMB_WRITE_FAILURE
Write failure

HAL_MMC_ERROR_RPMB_READ_FAILURE
Read failure

HAL_MMC_ERROR_RPMB_KEY_NOT_YET_PROG
Authentication Key not yet programmed

HAL_MMC_ERROR_RPMB_COUNTER_EXPIRED
Write Counter has expired i.e. reached its max value

HAL_MMC_ERROR_INVALID_CALLBACK
Invalid callback error

MMC context enumeration

MMC_CONTEXT_NONE
None

MMC_CONTEXT_READ_SINGLE_BLOCK
Read single block operation

MMC_CONTEXT_READ_MULTIPLE_BLOCK
Read multiple blocks operation

MMC_CONTEXT_WRITE_SINGLE_BLOCK
Write single block operation

MMC_CONTEXT_WRITE_MULTIPLE_BLOCK
Write multiple blocks operation

MMC_CONTEXT_IT
Process in Interrupt mode

MMC_CONTEXT_DMA
Process in DMA mode

MMC Voltage mode

MMC_HIGH_VOLTAGE_RANGE
High voltage in byte mode

MMC_DUAL_VOLTAGE_RANGE
Dual voltage in byte mode

MMC_LOW_VOLTAGE_RANGE
Low voltage in byte mode

EMMC_HIGH_VOLTAGE_RANGE
High voltage in sector mode

EMMC_DUAL_VOLTAGE_RANGE
Dual voltage in sector mode

EMMC_LOW_VOLTAGE_RANGE
Low voltage in sector mode

MMC_INVALID_VOLTAGE_RANGE

MMC Memory Cards

MMC_LOW_CAPACITY_CARD
MMC Card Capacity <=2Gbytes

MMC_HIGH_CAPACITY_CARD
MMC Card Capacity >2Gbytes and <2Tbytes

MMC Erase Type

HAL_MMC_ERASE
Erase the erase groups identified by CMD35 & 36

HAL_MMC_TRIM
Erase the write blocks identified by CMD35 & 36

HAL_MMC_DISCARD
Discard the write blocks identified by CMD35 & 36

HAL_MMC_SECURE_ERASE

Perform a secure purge according SRT on the erase groups identified by CMD35 & 36

HAL_MMC_SECURE_TRIM_STEP1

Mark the write blocks identified by CMD35 & 36 for secure erase

HAL_MMC_SECURE_TRIM_STEP2

Perform a secure purge according SRT on the write blocks previously identified

IS_MMC_ERASE_TYPE

MMC Secure Removal Type

HAL_MMC_SRT_ERASE

Information removed by an erase

HAL_MMC_SRT_WRITE_CHAR_ERASE

Information removed by an overwriting with a character followed by an erase

HAL_MMC_SRT_WRITE_CHAR_COMPL_RANDOM

Information removed by an overwriting with a character, its complement then a random character

HAL_MMC_SRT_VENDOR_DEFINED

Information removed using a vendor defined

IS_MMC_SRT_TYPE

MMC Partitions types

HAL_MMC_USER_AREA_PARTITION

User area partition

HAL_MMC_BOOT_PARTITION1

Boot partition 1

HAL_MMC_BOOT_PARTITION2

Boot partition 2

HAL_MMC_RPMB_PARTITION

RPMB partition

Exported Constants

MMC_BLOCKSIZE

Block size is 512 bytes

MMC Exported Macros

__HAL_MMC_RESET_HANDLE_STATE

Description:

- Reset MMC handle state.

Parameters:

- __HANDLE__: MMC Handle.

Return value:

- None

__HAL_MMC_ENABLE_IT

Description:

- Enable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_DISABLE_IT

Description:

- Disable the MMC device interrupt.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_MMC_GET_FLAG

Description:

- Check whether the specified MMC flag is set or not.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of MMC FLAG (SET or RESET).

__HAL_MMC_CLEAR_FLAG

Description:

- Clear the MMC's pending flags.

Parameters:

- __HANDLE__: MMC Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SD I/O interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_MMC_GET_IT

Description:

- Check whether the specified MMC interrupt has occurred or not.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of MMC IT (SET or RESET).

__HAL_MMC_CLEAR_IT

Description:

- Clear the MMC's interrupt pending bits.

Parameters:

- __HANDLE__: MMC Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SD I/O interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

MMC Card State enumeration structure

HAL_MMC_CARD_IDLE

Card is in idle state (can't be checked by CMD13)

HAL_MMC_CARD_READY

Card state is ready (can't be checked by CMD13)

HAL_MMC_CARD_IDENTIFICATION

Card is in identification state (can't be checked by CMD13)

HAL_MMC_CARD_STANDBY

Card is in standby state

HAL_MMC_CARD_TRANSFER

Card is in transfer state

HAL_MMC_CARD_SENDING

Card is sending an operation

HAL_MMC_CARD_RECEIVING

Card is receiving operation information

HAL_MMC_CARD_PROGRAMMING

Card is in programming state

HAL_MMC_CARD_DISCONNECTED

Card is disconnected

HAL_MMC_CARD_BUSTEST

Card is in bus test state

HAL_MMC_CARD_SLEEP

Card is in sleep state (can't be checked by CMD13)

HAL_MMC_CARD_ERROR

Card response Error (can't be checked by CMD13)

MMC Handle Structure definition

MMC_InitTypeDef

MMC_TypeDef

50 HAL MMC Extension Driver

50.1 MMCEx Firmware driver API description

The following section lists the various functions of the MMCEx library.

50.1.1 How to use this driver

The MMC Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_MMCEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_MMCEx_ReadBlocksDMAMultiBuffer() and HAL_MMCEx_WriteBlocksDMAMultiBuffer() functions.

50.1.2 Linked List management functions

This subsection provides a set of functions allowing to manage the needed functions.

This section contains the following APIs:

- *HAL_MMCEx_DMALinkedList_BuildNode()*
- *HAL_MMCEx_DMALinkedList_InsertNode()*
- *HAL_MMCEx_DMALinkedList_RemoveNode()*
- *HAL_MMCEx_DMALinkedList_LockNode()*
- *HAL_MMCEx_DMALinkedList_UnlockNode()*
- *HAL_MMCEx_DMALinkedList_EnableCircularMode()*
- *HAL_MMCEx_DMALinkedList_DisableCircularMode()*
- *HAL_MMCEx_DMALinkedList_ReadBlocks()*
- *HAL_MMCEx_DMALinkedList_WriteBlocks()*
- *HAL_MMCEx_DMALinkedList_ReadBlocks()*
- *HAL_MMCEx_DMALinkedList_WriteBlocks()*
- *HAL_MMCEx_DMALinkedList_EnableCircularMode()*
- *HAL_MMCEx_DMALinkedList_DisableCircularMode()*
- *HAL_MMCEx_Read_DMALnkLstBufCpltCallback()*
- *HAL_MMCEx_Write_DMALnkLstBufCpltCallback()*

50.1.3 Detailed description of functions

HAL_MMCEx_DMALinkedList_ReadBlocks

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_ReadBlocks (MMC_HandleTypeDef * hmmc, SDMMC_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMCEx_DMALinkedList_WriteBlocks

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_WriteBlocks (MMC_HandleTypeDef * hmmc, SDMMC_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_MMCEx_DMALinkedList_BuildNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_BuildNode (MMC_DMALinkNodeTypeDef * pNode, MMC_DMALinkNodeConfTypeDef * pNodeConf)

Function description

Build Linked List node.

Parameters

- **pNode:** Pointer to new node to add.
- **pNodeConf:** Pointer to configuration parameters for new node to add.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_InsertNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_InsertNode (MMC_DMALinkedListTypeDef * pLinkedList, MMC_DMALinkNodeTypeDef * pPrevNode, MMC_DMALinkNodeTypeDef * pNewNode)

Function description

Insert Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pPrevNode:** Pointer to previous node.
- **pNewNode:** Pointer to new node to insert.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_RemoveNode (MMC_DMALinkedListTypeDef * pLinkedList, MMC_DMALinkNodeTypeDef * pNode)

Function description

Remove Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_LockNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_LockNode (MMC_DMALinkNodeTypeDef * pNode)

Function description

Lock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_UnlockNode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_UnlockNode (MMC_DMALinkNodeTypeDef * pNode)

Function description

Unlock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_MMCEx_DMALinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_EnableCircularMode (SDMMC_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_MMCEx_DMALinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_MMCEx_DMALinkedList_DisableCircularMode (SDMMC_DMALinkedListTypeDef * pLinkedList)

Function description

HAL_MMCEx_Read_DMALnkLstBufCpltCallback

Function name

void HAL_MMCEx_Read_DMALnkLstBufCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

HAL_MMCEx_Write_DMALnkLstBufCpltCallback

Function name

void HAL_MMCEx_Write_DMALnkLstBufCpltCallback (MMC_HandleTypeDef * hmmc)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hmmc:** MMC handle

Return values

- **None:**

50.2 MMCEx Firmware driver defines

The following section lists the various define and macros of the module.

50.2.1 MMCEx

MMCEx

Linked List Wrapper

MMC_DMALinkNodeTypeDef

MMC_DMALinkNodeConfTypeDef

MMC_DMALinkedListTypeDef

HAL_MMCx_DMALinkedList_WriteCpltCallback

HAL_MMCx_DMALinkedList_ReadCpltCallback

51 HAL NAND Generic Driver

51.1 NAND Firmware driver registers structures

51.1.1 NAND_IDTypeDef

NAND_IDTypeDef is defined in the stm32u5xx_hal_nand.h

Data Fields

- ***uint8_t Maker_Id***
- ***uint8_t Device_Id***
- ***uint8_t Third_Id***
- ***uint8_t Fourth_Id***

Field Documentation

- ***uint8_t NAND_IDTypeDef::Maker_Id***
- ***uint8_t NAND_IDTypeDef::Device_Id***
- ***uint8_t NAND_IDTypeDef::Third_Id***
- ***uint8_t NAND_IDTypeDef::Fourth_Id***

51.1.2 NAND_AddressTypeDef

NAND_AddressTypeDef is defined in the stm32u5xx_hal_nand.h

Data Fields

- ***uint16_t Page***
- ***uint16_t Plane***
- ***uint16_t Block***

Field Documentation

- ***uint16_t NAND_AddressTypeDef::Page***
  NAND memory Page address
- ***uint16_t NAND_AddressTypeDef::Plane***
  NAND memory Zone address
- ***uint16_t NAND_AddressTypeDef::Block***
  NAND memory Block address

51.1.3 NAND_DeviceConfigTypeDef

NAND_DeviceConfigTypeDef is defined in the stm32u5xx_hal_nand.h

Data Fields

- ***uint32_t PageSize***
- ***uint32_t SpareAreaSize***
- ***uint32_t BlockSize***
- ***uint32_t BlockNbr***
- ***uint32_t PlaneNbr***
- ***uint32_t PlaneSize***
- ***FunctionalState ExtraCommandEnable***

Field Documentation

- ***uint32_t NAND_DeviceConfigTypeDef::PageSize***
  NAND memory page (without spare area) size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::SpareAreaSize***
  NAND memory spare area size measured in bytes for 8 bits addressing or words for 16 bits addressing
- ***uint32_t NAND_DeviceConfigTypeDef::BlockSize***
  NAND memory block size measured in number of pages

- ***uint32_t NAND_DeviceConfigTypeDef::BlockNbr***
  NAND memory number of total blocks
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneNbr***
  NAND memory number of planes
- ***uint32_t NAND_DeviceConfigTypeDef::PlaneSize***
  NAND memory zone size measured in number of blocks
- ***FunctionalState NAND_DeviceConfigTypeDef::ExtraCommandEnable***
  NAND extra command needed for Page reading mode. This parameter is mandatory for some NAND parts after the read command (NAND_CMD_AREA_TRUE1) and before DATA reading sequence. This parameter could be ENABLE or DISABLE Please check the Read Mode sequence in the NAND device datasheet

51.1.4 __NAND_HandleTypeDef

__NAND_HandleTypeDef is defined in the stm32u5xx_hal_nand.h

Data Fields

- ***FMC_NAND_TypeDef * Instance***
- ***FMC_NAND_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NAND_StateTypeDef State***
- ***NAND_DeviceConfigTypeDef Config***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* ItCallback***

Field Documentation

- ***FMC_NAND_TypeDef* __NAND_HandleTypeDef::Instance***
  Register base address
- ***FMC_NAND_InitTypeDef __NAND_HandleTypeDef::Init***
  NAND device control configuration parameters
- ***HAL_LockTypeDef __NAND_HandleTypeDef::Lock***
  NAND locking object
- ***__IO HAL_NAND_StateTypeDef __NAND_HandleTypeDef::State***
  NAND device access state
- ***NAND_DeviceConfigTypeDef __NAND_HandleTypeDef::Config***
  NAND physical characteristic information structure
- ***void(* __NAND_HandleTypeDef::MspInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp Init callback
- ***void(* __NAND_HandleTypeDef::MspDeInitCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND Msp DeInit callback
- ***void(* __NAND_HandleTypeDef::ItCallback)(struct __NAND_HandleTypeDef *hnand)***
  NAND IT callback

51.2 NAND Firmware driver API description

The following section lists the various functions of the NAND library.

51.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NAND flash memories. It uses the FMC layer functions to interface with NAND devices. This driver is used as follows:

- NAND flash memory configuration sequence using the function HAL_NAND_Init() with control and timing parameters for both common and attribute spaces.
- Read NAND flash memory maker and device IDs using the function HAL_NAND_Read_ID(). The read information is stored in the NAND_ID_TypeDef structure declared by the function caller.

- Access NAND flash memory by read/write operations using the functions HAL_NAND_Read_Page_8b()/ HAL_NAND_Read_SpareArea_8b(), HAL_NAND_Write_Page_8b()/HAL_NAND_Write_SpareArea_8b(), HAL_NAND_Read_Page_16b()/HAL_NAND_Read_SpareArea_16b(), HAL_NAND_Write_Page_16b()/ HAL_NAND_Write_SpareArea_16b() to read/write page(s)/spare area(s). These functions use specific device information (Block, page size..) predefined by the user in the NAND_DeviceConfigTypeDef structure. The read/write address information is contained by the Nand_Address_Typedef structure passed as parameter.
- Perform NAND flash Reset chip operation using the function HAL_NAND_Reset().
- Perform NAND flash erase block operation using the function HAL_NAND_Erase_Block(). The erase block address information is contained in the Nand_Address_Typedef structure passed as parameter.
- Read the NAND flash status operation using the function HAL_NAND_Read_Status().
- You can also control the NAND device by calling the control APIs HAL_NAND_ECC_Enable()/ HAL_NAND_ECC_Disable() to respectively enable/disable the ECC code correction feature or the function HAL_NAND_GetECC() to get the ECC correction code.
- You can monitor the NAND device HAL state by calling the function HAL_NAND_GetState()

Note: This driver is a set of generic APIs which handle standard NAND flash operations. If a NAND flash device contains different operations and/or implementations, it should be implemented separately.

Callback registration

The compilation define USE_HAL_NAND_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NAND_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NAND_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : NAND MspInit.
- MspDeInitCallback : NAND MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NAND_Init and if the state is HAL_NAND_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_NAND_Init and HAL_NAND_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NAND_Init and HAL_NAND_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NAND_RegisterCallback before calling HAL_NAND_DeInit or HAL_NAND_Init function. When The compilation define USE_HAL_NAND_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

51.2.2 NAND Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the NAND memory

This section contains the following APIs:

- *HAL_NAND_Init()*
- *HAL_NAND_DeInit()*
- *HAL_NAND_MspInit()*
- *HAL_NAND_MspDeInit()*
- *HAL_NAND_IRQHandler()*
- *HAL_NAND_ITCallback()*
- *HAL_NAND_ConfigDevice()*
- *HAL_NAND_Read_ID()*

51.2.3 NAND Input and Output functions

This section provides functions allowing to use and control the NAND memory

This section contains the following APIs:
- *HAL_NAND_Read_ID()*
- *HAL_NAND_Reset()*
- *HAL_NAND_ConfigDevice()*
- *HAL_NAND_Read_Page_8b()*
- *HAL_NAND_Read_Page_16b()*
- *HAL_NAND_Write_Page_8b()*
- *HAL_NAND_Write_Page_16b()*
- *HAL_NAND_Read_SpareArea_8b()*
- *HAL_NAND_Read_SpareArea_16b()*
- *HAL_NAND_Write_SpareArea_8b()*
- *HAL_NAND_Write_SpareArea_16b()*
- *HAL_NAND_Erase_Block()*
- *HAL_NAND_Address_Inc()*
- *HAL_NAND_RegisterCallback()*
- *HAL_NAND_UnRegisterCallback()*

51.2.4 NAND Control functions

This subsection provides a set of functions allowing to control dynamically the NAND interface.

This section contains the following APIs:
- *HAL_NAND_ECC_Enable()*
- *HAL_NAND_ECC_Disable()*
- *HAL_NAND_GetECC()*

51.2.5 NAND State functions

This subsection permits to get in run-time the status of the NAND controller and the data flow.

This section contains the following APIs:
- *HAL_NAND_GetState()*
- *HAL_NAND_Read_Status()*

51.2.6 Detailed description of functions

HAL_NAND_Init

Function name

HAL_StatusTypeDef HAL_NAND_Init (NAND_HandleTypeDef * hnand, FMC_NAND_PCC_TimingTypeDef * ComSpace_Timing, FMC_NAND_PCC_TimingTypeDef * AttSpace_Timing)

Function description

Perform NAND memory Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ComSpace_Timing:** pointer to Common space timing structure
- **AttSpace_Timing:** pointer to Attribute space timing structure

Return values

- **HAL:** status

HAL_NAND_DeInit

Function name

HAL_StatusTypeDef HAL_NAND_DeInit (NAND_HandleTypeDef * hnand)

Function description

Perform NAND memory De-Initialization sequence.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ConfigDevice

Function name

HAL_StatusTypeDef HAL_NAND_ConfigDevice (NAND_HandleTypeDef * hnand, NAND_DeviceConfigTypeDef * pDeviceConfig)

Function description

Configure the device: Enter the physical parameters of the device.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pDeviceConfig:** pointer to NAND_DeviceConfigTypeDef structure

Return values

- **HAL:** status

HAL_NAND_Read_ID

Function name

HAL_StatusTypeDef HAL_NAND_Read_ID (NAND_HandleTypeDef * hnand, NAND_IDTypeDef * pNAND_ID)

Function description

Read the NAND memory electronic signature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pNAND_ID:** NAND ID structure

Return values

- **HAL:** status

HAL_NAND_MspInit

Function name

void HAL_NAND_MspInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP Init.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_MspDeInit

Function name

void HAL_NAND_MspDeInit (NAND_HandleTypeDef * hnand)

Function description

NAND MSP DeInit.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_IRQHandler

Function name

void HAL_NAND_IRQHandler (NAND_HandleTypeDef * hnand)

Function description

This function handles NAND device interrupt request.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ITCallback

Function name

void HAL_NAND_ITCallback (NAND_HandleTypeDef * hnand)

Function description

NAND interrupt feature callback.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **None:**

HAL_NAND_Reset

Function name

HAL_StatusTypeDef HAL_NAND_Reset (NAND_HandleTypeDef * hnand)

Function description

NAND memory reset.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_Read_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint8_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint8_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_8b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_8b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint8_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (8-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Read_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_Page_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumPageToRead)

Function description

Read Page(s) from NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to destination read buffer. pBuffer should be 16bits aligned
- **NumPageToRead:** number of pages to read from block

Return values

- **HAL:** status

HAL_NAND_Write_Page_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_Page_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint16_t * pBuffer, uint32_t NumPageToWrite)

Function description

Write Page(s) to NAND memory block (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned
- **NumPageToWrite:** number of pages to write to block

Return values

- **HAL:** status

HAL_NAND_Read_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Read_SpareArea_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, uint16_t * pBuffer, uint32_t NumSpareAreaToRead)

Function description

Read Spare area(s) from NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaToRead:** Number of spare area to read

Return values

- **HAL:** status

HAL_NAND_Write_SpareArea_16b

Function name

HAL_StatusTypeDef HAL_NAND_Write_SpareArea_16b (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress, const uint16_t * pBuffer, uint32_t NumSpareAreaTowrite)

Function description

Write Spare area(s) to NAND memory (16-bits addressing)

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure
- **pBuffer:** pointer to source buffer to write. pBuffer should be 16bits aligned.
- **NumSpareAreaTowrite:** number of spare areas to write to block

Return values

- **HAL:** status

HAL_NAND_Erase_Block

Function name

HAL_StatusTypeDef HAL_NAND_Erase_Block (NAND_HandleTypeDef * hnand, const NAND_AddressTypeDef * pAddress)

Function description

NAND memory Block erase.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **HAL:** status

HAL_NAND_Address_Inc

Function name

uint32_t HAL_NAND_Address_Inc (const NAND_HandleTypeDef * hnand, NAND_AddressTypeDef * pAddress)

Function description

Increment the NAND memory address.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **pAddress:** pointer to NAND address structure

Return values

- **The:** new status of the increment address operation. It can be:
  – NAND_VALID_ADDRESS: When the new address is valid address
  – NAND_INVALID_ADDRESS: When the new address is invalid address

HAL_NAND_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_RegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId, pNAND_CallbackTypeDef pCallback)

Function description

Register a User NAND Callback To be used to override the weak predefined callback.

Parameters

- **hnand:** : NAND handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  – HAL_NAND_MSP_INIT_CB_ID NAND MspInit callback ID
  – HAL_NAND_MSP_DEINIT_CB_ID NAND MspDeInit callback ID
  – HAL_NAND_IT_CB_ID NAND IT callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_NAND_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NAND_UnRegisterCallback (NAND_HandleTypeDef * hnand, HAL_NAND_CallbackIDTypeDef CallbackId)

Function description

Unregister a User NAND Callback NAND Callback is redirected to the weak predefined callback.

Parameters

- **hnand:** : NAND handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_NAND_MSP_INIT_CB_ID NAND MspInit callback ID
  – HAL_NAND_MSP_DEINIT_CB_ID NAND MspDeInit callback ID
  – HAL_NAND_IT_CB_ID NAND IT callback ID

Return values

- **status:**

HAL_NAND_ECC_Enable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Enable (NAND_HandleTypeDef * hnand)

Function description

Enables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_ECC_Disable

Function name

HAL_StatusTypeDef HAL_NAND_ECC_Disable (NAND_HandleTypeDef * hnand)

Function description

Disables dynamically FMC_NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** status

HAL_NAND_GetECC

Function name

HAL_StatusTypeDef HAL_NAND_GetECC (NAND_HandleTypeDef * hnand, uint32_t * ECCval, uint32_t Timeout)

Function description

Disables dynamically NAND ECC feature.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.
- **ECCval:** pointer to ECC value
- **Timeout:** maximum timeout to wait

Return values

- **HAL:** status

HAL_NAND_GetState

Function name

HAL_NAND_StateTypeDef HAL_NAND_GetState (const NAND_HandleTypeDef * hnand)

Function description

return the NAND state

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **HAL:** state

HAL_NAND_Read_Status

Function name

uint32_t HAL_NAND_Read_Status (const NAND_HandleTypeDef * hnand)

Function description

NAND memory read status.

Parameters

- **hnand:** pointer to a NAND_HandleTypeDef structure that contains the configuration information for NAND module.

Return values

- **NAND:** status

51.3 NAND Firmware driver defines

The following section lists the various define and macros of the module.

51.3.1 NAND

NAND

NAND Exported Macros

__HAL_NAND_RESET_HANDLE_STATE

Description:

- Reset NAND handle state.

Parameters:

- __HANDLE__: specifies the NAND handle.

Return value:

- None

52 HAL NOR Generic Driver

52.1 NOR Firmware driver registers structures

52.1.1 NOR_IDTypeDef

NOR_IDTypeDef is defined in the stm32u5xx_hal_nor.h

Data Fields

- ***uint16_t Manufacturer_Code***
- ***uint16_t Device_Code1***
- ***uint16_t Device_Code2***
- ***uint16_t Device_Code3***

Field Documentation

- ***uint16_t NOR_IDTypeDef::Manufacturer_Code***
  Defines the device's manufacturer code used to identify the memory
- ***uint16_t NOR_IDTypeDef::Device_Code1***
- ***uint16_t NOR_IDTypeDef::Device_Code2***
- ***uint16_t NOR_IDTypeDef::Device_Code3***
  Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set.They can also be accessed by issuing an Auto Select command

52.1.2 NOR_CFITypeDef

NOR_CFITypeDef is defined in the stm32u5xx_hal_nor.h

Data Fields

- ***uint16_t CFI_1***
- ***uint16_t CFI_2***
- ***uint16_t CFI_3***
- ***uint16_t CFI_4***

Field Documentation

- ***uint16_t NOR_CFITypeDef::CFI_1***
  < Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- ***uint16_t NOR_CFITypeDef::CFI_2***
- ***uint16_t NOR_CFITypeDef::CFI_3***
- ***uint16_t NOR_CFITypeDef::CFI_4***

52.1.3 __NOR_HandleTypeDef

__NOR_HandleTypeDef is defined in the stm32u5xx_hal_nor.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***
- ***uint32_t CommandSet***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __NOR_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __NOR_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __NOR_HandleTypeDef::Init***
  NOR device control configuration parameters
- ***HAL_LockTypeDef __NOR_HandleTypeDef::Lock***
  NOR locking object
- ***__IO HAL_NOR_StateTypeDef __NOR_HandleTypeDef::State***
  NOR device access state
- ***uint32_t __NOR_HandleTypeDef::CommandSet***
  NOR algorithm command set and control
- ***void(* __NOR_HandleTypeDef::MspInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp Init callback
- ***void(* __NOR_HandleTypeDef::MspDeInitCallback)(struct __NOR_HandleTypeDef *hnor)***
  NOR Msp DeInit callback

52.2 NOR Firmware driver API description

The following section lists the various functions of the NOR library.

52.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function HAL_NOR_Init() with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function HAL_NOR_Read_ID(). The read information is stored in the NOR_ID_TypeDef structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions HAL_NOR_Read(), HAL_NOR_Program().
- Perform NOR flash erase block/chip operations using the functions HAL_NOR_Erase_Block() and HAL_NOR_Erase_Chip().
- Read the NOR flash CFI (common flash interface) IDs using the function HAL_NOR_Read_CFI(). The read information is stored in the NOR_CFI_TypeDef structure declared by the function caller.
- You can also control the NOR device by calling the control APIs HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable() to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function HAL_NOR_GetState()

Note: *This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.*

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- NOR_WRITE : NOR memory write data to specified address

Callback registration

The compilation define USE_HAL_NOR_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_NOR_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : NOR MspInit.
- MspDeInitCallback : NOR MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_NOR_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : NOR MspInit.

- MspDeInitCallback : NOR MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_NOR_Init and if the state is HAL_NOR_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_NOR_Init and HAL_NOR_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_NOR_Init and HAL_NOR_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_NOR_RegisterCallback before calling HAL_NOR_DeInit or HAL_NOR_Init function. When The compilation define USE_HAL_NOR_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

52.2.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- *HAL_NOR_Init()*
- *HAL_NOR_DeInit()*
- *HAL_NOR_MspInit()*
- *HAL_NOR_MspDeInit()*
- *HAL_NOR_MspWait()*

52.2.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- *HAL_NOR_Read_ID()*
- *HAL_NOR_ReturnToReadMode()*
- *HAL_NOR_Read()*
- *HAL_NOR_Program()*
- *HAL_NOR_ReadBuffer()*
- *HAL_NOR_ProgramBuffer()*
- *HAL_NOR_Erase_Block()*
- *HAL_NOR_Erase_Chip()*
- *HAL_NOR_Read_CFI()*
- *HAL_NOR_RegisterCallback()*
- *HAL_NOR_UnRegisterCallback()*

52.2.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- *HAL_NOR_WriteOperation_Enable()*
- *HAL_NOR_WriteOperation_Disable()*

52.2.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- *HAL_NOR_GetState()*
- *HAL_NOR_GetStatus()*

52.2.6 Detailed description of functions

HAL_NOR_Init

Function name

HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Perform the NOR memory Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timing:** pointer to NOR control timing structure
- **ExtTiming:** pointer to NOR extended mode timing structure

Return values

- **HAL:** status

HAL_NOR_DeInit

Function name

HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)

Function description

Perform NOR memory De-Initialization sequence.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_MspInit

Function name

void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP Init.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspDeInit

Function name

void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)

Function description

NOR MSP DeInit.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **None:**

HAL_NOR_MspWait

Function name

void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)

Function description

NOR MSP Wait for Ready/Busy signal.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Timeout:** Maximum timeout value

Return values

- **None:**

HAL_NOR_Read_ID

Function name

HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)

Function description

Read NOR flash IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_ID:** pointer to NOR ID structure

Return values

- **HAL:** status

HAL_NOR_ReturnToReadMode

Function name

HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)

Function description

Returns the NOR memory to Read mode.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_Read

Function name

HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Read data from NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** pointer to Device address
- **pData:** pointer to read data

Return values

- **HAL:** status

HAL_NOR_Program

Function name

HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)

Function description

Program data to NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pAddress:** Device address
- **pData:** pointer to the data to write

Return values

- **HAL:** status

HAL_NOR_ReadBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Reads a half-word buffer from the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** number of Half word to read.

Return values

- **HAL:** status

HAL_NOR_ProgramBuffer

Function name

HAL_StatusTypeDef HAL_NOR_ProgramBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function description

Writes a half-word buffer to the NOR memory.

Parameters

- **hnor:** pointer to the NOR handle
- **uwAddress:** NOR memory internal start write address
- **pData:** pointer to source data buffer.
- **uwBufferSize:** Size of the buffer to write

Return values

- **HAL:** status

HAL_NOR_Erase_Block

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Block (NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)

Function description

Erase the specified block of the NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** Block to erase address
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Erase_Chip

Function name

HAL_StatusTypeDef HAL_NOR_Erase_Chip (NOR_HandleTypeDef * hnor, uint32_t Address)

Function description

Erase the entire NOR chip.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address

Return values

- **HAL:** status

HAL_NOR_Read_CFI

Function name

HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)

Function description

Read NOR flash CFI IDs.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **pNOR_CFI:** pointer to NOR CFI IDs structure

Return values

- **HAL:** status

HAL_NOR_RegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_RegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId, pNOR_CallbackTypeDef pCallback)

Function description

Register a User NOR Callback To be used to override the weak predefined callback.

Parameters

- **hnor:** : NOR handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_NOR_MSP_INIT_CB_ID NOR MspInit callback ID
  - HAL_NOR_MSP_DEINIT_CB_ID NOR MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_NOR_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_NOR_UnRegisterCallback (NOR_HandleTypeDef * hnor, HAL_NOR_CallbackIDTypeDef CallbackId)

Function description

Unregister a User NOR Callback NOR Callback is redirected to the weak predefined callback.

Parameters

- **hnor:** : NOR handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_NOR_MSP_INIT_CB_ID NOR MspInit callback ID
  - HAL_NOR_MSP_DEINIT_CB_ID NOR MspDeInit callback ID

Return values

- **status:**

HAL_NOR_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)

Function description

Enables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)

Function description

Disables dynamically NOR write operation.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **HAL:** status

HAL_NOR_GetState

Function name

HAL_NOR_StateTypeDef HAL_NOR_GetState (const NOR_HandleTypeDef * hnor)

Function description

return the NOR controller state

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.

Return values

- **NOR:** controller state

HAL_NOR_GetStatus

Function name

HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)

Function description

Returns the NOR operation status.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** Device address
- **Timeout:** NOR programming Timeout

Return values

- **NOR_Status:** The returned value can be: HAL_NOR_STATUS_SUCCESS, HAL_NOR_STATUS_ERROR or HAL_NOR_STATUS_TIMEOUT

52.3 NOR Firmware driver defines

The following section lists the various define and macros of the module.

52.3.1 NOR

NOR

NOR Exported Macros

__HAL_NOR_RESET_HANDLE_STATE

Description:

- Reset NOR handle state.

Parameters:

- __HANDLE__: specifies the NOR handle.

Return value:

- None

53 HAL OPAMP Generic Driver

53.1 OPAMP Firmware driver registers structures

53.1.1 OPAMP_InitTypeDef

OPAMP_InitTypeDef is defined in the stm32u5xx_hal_opamp.h

Data Fields

- ***uint32_t PowerSupplyRange***
- ***uint32_t PowerMode***
- ***uint32_t Mode***
- ***uint32_t InvertingInput***
- ***uint32_t NonInvertingInput***
- ***uint32_t PgaGain***
- ***uint32_t UserTrimming***
- ***uint32_t TrimmingValueP***
- ***uint32_t TrimmingValueN***
- ***uint32_t TrimmingValuePLowPower***
- ***uint32_t TrimmingValueNLowPower***

Field Documentation

- ***uint32_t OPAMP_InitTypeDef::PowerSupplyRange***
  Specifies the power supply range: above or under 2.4V. This parameter is not used, but kept for compatibility with other families. Caution: Whatever this parameter, the effective PowerSupplyRange is forced to OPAMP_POWERSUPPLY_HIGH in HAL_OPAMP_Init. This parameter applies to the both OPAMP instances.
- ***uint32_t OPAMP_InitTypeDef::PowerMode***
  Specifies the power mode Normal or Low-Power and the speed mode Normal or High. This parameter must be a value of ***OPAMP_PowerMode***
- ***uint32_t OPAMP_InitTypeDef::Mode***
  Specifies the OPAMP mode (Standalone, Follower or PGA) This parameter must be a value of ***OPAMP_Mode***
- ***uint32_t OPAMP_InitTypeDef::InvertingInput***
  Specifies the inverting input in Standalone & PGA modes. In Follower mode this parameter is Not Applicable. This parameter must be a value of ***OPAMP_InvertingInput***
- ***uint32_t OPAMP_InitTypeDef::NonInvertingInput***
  Specifies the non inverting input of the opamp. This parameter must be a value of ***OPAMP_NonInvertingInput***
- ***uint32_t OPAMP_InitTypeDef::PgaGain***
  Specifies the gain (2, 4, 8 or 16) in PGA mode i.e. when mode is OPAMP_PGA_MODE. This parameter must be a value of ***OPAMP_PgaGain***
- ***uint32_t OPAMP_InitTypeDef::UserTrimming***
  Specifies the trimming mode is either factory or user trimming This parameter must be a value of ***OPAMP_UserTrimming***.
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***
  Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***
  Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValuePLowPower***
  Specifies the offset trimming value (PMOS) in lowpower mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value

- ***uint32_t OPAMP_InitTypeDef::TrimmingValueNLowPower***
  Specifies the offset trimming value (NMOS) in lowpower mode i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 31 16 is typical default value

53.1.2 __OPAMP_HandleTypeDef

__OPAMP_HandleTypeDef is defined in the stm32u5xx_hal_opamp.h

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OPAMP_TypeDef* __OPAMP_HandleTypeDef::Instance***
  OPAMP instance's registers base address
- ***OPAMP_InitTypeDef __OPAMP_HandleTypeDef::Init***
  OPAMP required parameters
- ***HAL_StatusTypeDef __OPAMP_HandleTypeDef::Status***
  OPAMP peripheral status
- ***HAL_LockTypeDef __OPAMP_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_OPAMP_StateTypeDef __OPAMP_HandleTypeDef::State***
  OPAMP communication state
- ***void(* __OPAMP_HandleTypeDef::MspInitCallback)(struct __OPAMP_HandleTypeDef *hopamp)***
- ***void(* __OPAMP_HandleTypeDef::MspDeInitCallback)(struct __OPAMP_HandleTypeDef *hopamp)***

53.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

53.2.1 OPAMP Peripheral Features

The device integrates 1 or 2 operational amplifiers OPAMP1 & OPAMP2

1. The OPAMP(s) provide(s) several exclusive running modes.
   - Standalone mode
   - Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   - Follower mode
2. Each OPAMP(s) can be configured in normal and low power mode with different speeds.
3. The OPAMP(s) provide(s) calibration capabilities.
   - Calibration aims at correcting some offset for running mode.
   - The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
   - The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
   - HAL_OPAMP_SelfCalibrate:
     - Runs automatically the calibration.
     - Enables the user trimming mode
     - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
     - HAL_OPAMPEx_SelfCalibrateAll runs calibration of all OPAMPs in parallel to save search time.

4. Running mode: Standalone mode
   – Gain is set externally (gain depends on external loads).
   – Follower mode also possible externally by connecting the inverting input to the output.
5. Running mode: Follower mode
   – No Inverting Input is connected.
6. Running mode: Programmable Gain Amplifier (PGA) mode (Resistor feedback output)
   – The OPAMP(s) output(s) can be internally connected to resistor feedback output.
   – OPAMP gain is either 2, 4, 8 or 16.
7. The OPAMPs inverting input can be selected according to the Reference Manual "OPAMP function description" chapter.
8. The OPAMPs non inverting input can be selected according to the Reference Manual "OPAMP function description" chapter.

53.2.2 How to use this driver

Speed & power mode

To run in low power mode with different speed:
1. Configure the OPAMP using HAL_OPAMP_Init() function:
   – Select OPAMP_POWERMODE_LOWPOWER_NORMALSPEED
   – Select OPAMP_POWERMODE_LOWPOWER_HIGHSPEED
   – Select OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED
   – Select OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED

Calibration

To run the OPAMP calibration self calibration:
1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the OPAMP, perform the following steps:
1. Fill in the HAL_OPAMP_MspInit() to
   – Enable the OPAMP Peripheral clock using macro __HAL_RCC_OPAMP_CLK_ENABLE()
   – Configure the OPAMP input and output in analog mode using HAL_GPIO_Init() to map the OPAMP output to the GPIO pin.
2. Registrate Callbacks
   – The compilation define USE_HAL_OPAMP_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.
   – Use Functions @ref HAL_OPAMP_RegisterCallback() to register a user callback, it allows to register following callbacks:
     ◦ MspInitCallback : OPAMP MspInit.
     ◦ MspDeInitCallback : OPAMP MspFeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
   – Use function @ref HAL_OPAMP_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
     ◦ MspInitCallback : OPAMP MspInit.
     ◦ MspDeInitCallback : OPAMP MspdeInit.
     ◦ All Callbacks

3. Configure the OPAMP using HAL_OPAMP_Init() function:
   – Select the mode
   – Select the inverting input
   – Select the non-inverting input
   – If PGA mode is enabled, Select if inverting input is connected.
   – Select either factory or user defined trimming mode.
   – If the user-defined trimming mode is enabled, select PMOS & NMOS trimming values (typically values set by HAL_OPAMP_SelfCalibrate function).
4. Enable the OPAMP using HAL_OPAMP_Start() function.
5. Disable the OPAMP using HAL_OPAMP_Stop() function.
6. Lock the OPAMP in running mode using HAL_OPAMP_Lock() function. Caution: On STM32U5, HAL OPAMP lock is software lock only (not hardware lock as on some other STM32 devices)
7. If needed, unlock the OPAMP using HAL_OPAMPEx_Unlock() function.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, fill in the HAL_OPAMP_MspInit()
   – This is the case for instance if you wish to use new OPAMP I/O
2. Configure the OPAMP using HAL_OPAMP_Init() function:
   – As in configure case, select first the parameters you wish to modify.
3. Change from low power mode to normal power mode (& vice versa) requires first HAL_OPAMP_DeInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, of OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

53.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_OPAMP_Init()*
- *HAL_OPAMP_DeInit()*
- *HAL_OPAMP_MspInit()*
- *HAL_OPAMP_MspDeInit()*

53.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- *HAL_OPAMP_Start()*
- *HAL_OPAMP_Stop()*
- *HAL_OPAMP_SelfCalibrate()*

53.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- *HAL_OPAMP_Lock()*
- *HAL_OPAMP_GetTrimOffset()*

53.2.6 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OPAMP_GetState()*

53.2.7 Peripheral Callback functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *HAL_OPAMP_RegisterCallback()*
- *HAL_OPAMP_UnRegisterCallback()*

53.2.8 Extended IO operation functions

- OPAMP Self calibration.

This section contains the following APIs:

- *Section 53.2.9: HAL_OPAMPEx_SelfCalibrateAll()*

53.2.9 Detailed description of functions

HAL_OPAMP_Init

Function name

HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)

Function description

Initializes the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and initialize the associated handle.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

HAL_OPAMP_DeInit

Function name

HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize the OPAMP peripheral.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- Deinitialization can be performed if the OPAMP configuration is locked. (the lock is SW in U5)

HAL_OPAMP_MspInit

Function name

void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)

Function description

Initialize the OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_MspDeInit

Function name

void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)

Function description

DeInitialize OPAMP MSP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **None:**

HAL_OPAMP_Start

Function name

HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)

Function description

Start the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_Stop

Function name

HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)

Function description

Stop the OPAMP.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

HAL_OPAMP_SelfCalibrate

Function name

HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)

Function description

Run the self calibration of one OPAMP.

Parameters

- **hopamp:** handle

Return values

- **Updated:** offset trimming values (PMOS & NMOS), user trimming is enabled
- **HAL:** status

Notes

- Calibration is performed in the mode specified in OPAMP init structure (mode normal or low-power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.
- Calibration runs about 10 ms.

HAL_OPAMP_Lock

Function name

HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)

Function description

Lock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- On STM32U5, HAL OPAMP lock is software lock only (in contrast of hardware lock available on some other STM32 devices).

HAL_OPAMP_GetTrimOffset

Function name

HAL_OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (const OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)

Function description

Return the OPAMP factory trimming value.

Parameters

- **hopamp:** : OPAMP handle
- **trimmingoffset:** : Trimming offset (P or N) This parameter must be a value of OPAMP Factory Trimming

Return values

- **Trimming:** value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available

Notes

- On STM32U5 OPAMP, user can retrieve factory trimming if OPAMP has never been set to user trimming before. Therefore, this function must be called when OPAMP init parameter "UserTrimming" is set to trimming factory, and before OPAMP calibration (function "HAL_OPAMP_SelfCalibrate()"). Otherwise, factory trimming value cannot be retrieved and error status is returned.
- Calibration parameter retrieved is corresponding to the mode specified in OPAMP init structure (mode normal or low-power). To retrieve calibration parameters for both modes, repeat this function after OPAMP init structure accordingly updated.

HAL_OPAMP_GetState

Function name

HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (const OPAMP_HandleTypeDef * hopamp)

Function description

Return the OPAMP handle state.

Parameters

- **hopamp:** : OPAMP handle

Return values

- **HAL:** state

HAL_OPAMP_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OPAMP_RegisterCallback (OPAMP_HandleTypeDef * hopamp, HAL_OPAMP_CallbackIDTypeDef CallbackID, pOPAMP_CallbackTypeDef pCallback)

Function description

Register a User OPAMP Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hopamp:** : OPAMP handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  – HAL_OPAMP_MSP_INIT_CB_ID OPAMP MspInit callback ID
  – HAL_OPAMP_MSP_DEINIT_CB_ID OPAMP MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_OPAMP_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OPAMP_UnRegisterCallback (OPAMP_HandleTypeDef * hopamp, HAL_OPAMP_CallbackIDTypeDef CallbackID)

Function description

Unregister a User OPAMP Callback OPAMP Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hopamp:** : OPAMP handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_OPAMP_MSP_INIT_CB_ID OPAMP MSP Init Callback ID
  – HAL_OPAMP_MSP_DEINIT_CB_ID OPAMP MSP DeInit Callback ID
  – HAL_OPAMP_ALL_CB_ID OPAMP All Callbacks

Return values

- **status:**

53.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

53.3.1 OPAMP

OPAMP

OPAMP Exported Macros

__HAL_OPAMP_RESET_HANDLE_STATE

Description:

- Reset OPAMP handle state.

Parameters:

- __HANDLE__: OPAMP handle.

Return value:

- None

OPAMP Factory Trimming

OPAMP_FACTORYTRIMMING_DUMMY

Dummy value if trimming value could not be retrieved

OPAMP_FACTORYTRIMMING_N

Offset trimming N

OPAMP_FACTORYTRIMMING_P

Offset trimming P

OPAMP Inverting Input

OPAMP_INVERTINGINPUT_IO0

OPAMP inverting input connected to dedicated IO pin low-leakage

OPAMP_INVERTINGINPUT_IO1

OPAMP inverting input connected to alternative IO pin available on some device packages

OPAMP_INVERTINGINPUT_CONNECT_NO

OPAMP inverting input not connected externally (PGA mode only)

OPAMP Mode

OPAMP_STANDALONE_MODE

standalone mode

OPAMP_PGA_MODE

PGA mode

OPAMP_FOLLOWER_MODE

follower mode

OPAMP Non Inverting Input

OPAMP_NONINVERTINGINPUT_IO0

OPAMP non-inverting input connected to dedicated IO pin

OPAMP_NONINVERTINGINPUT_DAC_CH

OPAMP non-inverting input connected internally to DAC channel

OPAMP Pga Gain

OPAMP_PGA_GAIN_2

PGA gain = 2

OPAMP_PGA_GAIN_4

PGA gain = 4

OPAMP_PGA_GAIN_8

PGA gain = 8

OPAMP_PGA_GAIN_16

PGA gain = 16

OPAMP PowerMode

OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED

OPAMP power mode normal speed normal

OPAMP_POWERMODE_LOWPOWER_NORMALSPEED

OPAMP power mode low-power speed normal

OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED

OPAMP power mode normal speed high

OPAMP_POWERMODE_LOWPOWER_HIGHSPEED

OPAMP power mode low-power speed high

OPAMP PowerSupplyRange

OPAMP_POWERSUPPLY_LOW

Power supply range low (VDDA lower than 2.4V)

OPAMP_POWERSUPPLY_HIGH

Power supply range high (VDDA higher than 2.4V)

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY

Factory trimming

OPAMP_TRIMMING_USER

User trimming

54 HAL OPAMP Extension Driver

54.1 OPAMPEx Firmware driver API description

The following section lists the various functions of the OPAMPEx library.

54.1.1 Initialization and de-initialization functions

This section contains the following APIs:

- *Section 54.1.7: HAL_OPAMP_Init()*
- *Section 54.1.7: HAL_OPAMP_DeInit()*
- *Section 54.1.7: HAL_OPAMP_MspInit()*
- *Section 54.1.7: HAL_OPAMP_MspDeInit()*

54.1.2 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- *Section 54.1.7: HAL_OPAMP_Start()*
- *Section 54.1.7: HAL_OPAMP_Stop()*
- *Section 54.1.7: HAL_OPAMP_SelfCalibrate()*

54.1.3 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- *Section 54.1.7: HAL_OPAMP_Lock()*
- *Section 54.1.7: HAL_OPAMP_GetTrimOffset()*

54.1.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *Section 54.1.7: HAL_OPAMP_GetState()*

54.1.5 Peripheral Callback functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- *Section 54.1.7: HAL_OPAMP_RegisterCallback()*
- *Section 54.1.7: HAL_OPAMP_UnRegisterCallback()*

54.1.6 Extended IO operation functions

- OPAMP Self calibration.

This section contains the following APIs:

- *HAL_OPAMPEx_SelfCalibrateAll()*

54.1.7 Detailed description of functions

HAL_OPAMPEx_SelfCalibrateAll

Function name

HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1, OPAMP_HandleTypeDef * hopamp2)

Function description

Run the self calibration of the 2 OPAMPs in parallel.

Parameters

- **hopamp1:** handle
- **hopamp2:** handle

Return values

- **HAL:** status

Notes

- Trimming values (PMOS & NMOS) are updated and user trimming is enabled is calibration is successful.
- Calibration is performed in the mode specified in OPAMP init structure (mode normal or low-power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated.
- Calibration runs about 10 ms (5 dichotomy steps, repeated for P and N transistors: 10 steps with 1 ms for each step).

HAL_OPAMPEx_Unlock

Function name

HAL_StatusTypeDef HAL_OPAMPEx_Unlock (OPAMP_HandleTypeDef * hopamp)

Function description

Unlock the selected OPAMP configuration.

Parameters

- **hopamp:** OPAMP handle

Return values

- **HAL:** status

Notes

- This function must be called only when OPAMP is in state "locked".

55 HAL OSPI Generic Driver

55.1 OSPI Firmware driver registers structures

55.1.1 OSPI_InitTypeDef

OSPI_InitTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t FifoThreshold***
- ***uint32_t DualQuad***
- ***uint32_t MemoryType***
- ***uint32_t DeviceSize***
- ***uint32_t ChipSelectHighTime***
- ***uint32_t FreeRunningClock***
- ***uint32_t ClockMode***
- ***uint32_t WrapSize***
- ***uint32_t ClockPrescaler***
- ***uint32_t SampleShifting***
- ***uint32_t DelayHoldQuarterCycle***
- ***uint32_t ChipSelectBoundary***
- ***uint32_t DelayBlockBypass***
- ***uint32_t MaxTran***
- ***uint32_t Refresh***

Field Documentation

- ***uint32_t OSPI_InitTypeDef::FifoThreshold***
  This is the threshold used by the Peripheral to generate the interrupt indicating that data are available in reception or free place is available in transmission. This parameter can be a value between 1 and 32
- ***uint32_t OSPI_InitTypeDef::DualQuad***
  It enables or not the dual-quad mode which allow to access up to quad mode on two different devices to increase the throughput. This parameter can be a value of ***OSPI_DualQuad***
- ***uint32_t OSPI_InitTypeDef::MemoryType***
  It indicates the external device type connected to the OSPI. This parameter can be a value of ***OSPI_MemoryType***
- ***uint32_t OSPI_InitTypeDef::DeviceSize***
  It defines the size of the external device connected to the OSPI, it corresponds to the number of address bits required to access the external device. This parameter can be a value between 1 and 32
- ***uint32_t OSPI_InitTypeDef::ChipSelectHighTime***
  It defines the minimum number of clocks which the chip select must remain high between commands. This parameter can be a value between 1 and 8
- ***uint32_t OSPI_InitTypeDef::FreeRunningClock***
  It enables or not the free running clock. This parameter can be a value of ***OSPI_FreeRunningClock***
- ***uint32_t OSPI_InitTypeDef::ClockMode***
  It indicates the level of clock when the chip select is released. This parameter can be a value of ***OSPI_ClockMode***
- ***uint32_t OSPI_InitTypeDef::WrapSize***
  It indicates the wrap-size corresponding the external device configuration. This parameter can be a value of ***OSPI_WrapSize***
- ***uint32_t OSPI_InitTypeDef::ClockPrescaler***
  It specifies the prescaler factor used for generating the external clock based on the AHB clock. This parameter can be a value between 1 and 256
- ***uint32_t OSPI_InitTypeDef::SampleShifting***
  It allows to delay to 1/2 cycle the data sampling in order to take in account external signal delays. This parameter can be a value of ***OSPI_SampleShifting***

- ***uint32_t OSPI_InitTypeDef::DelayHoldQuarterCycle***
  It allows to hold to 1/4 cycle the data. This parameter can be a value of ***OSPI_DelayHoldQuarterCycle***
- ***uint32_t OSPI_InitTypeDef::ChipSelectBoundary***
  It enables the transaction boundary feature and defines the boundary of bytes to release the chip select. This parameter can be a value between 0 and 31
- ***uint32_t OSPI_InitTypeDef::DelayBlockBypass***
  It enables the delay block bypass, so the sampling is not affected by the delay block. This parameter can be a value of ***OSPI_DelayBlockBypass***
- ***uint32_t OSPI_InitTypeDef::MaxTran***
  It enables the communication regulation feature. The chip select is released every MaxTran+1 bytes when the other OctoSPI request the access to the bus. This parameter can be a value between 0 and 255
- ***uint32_t OSPI_InitTypeDef::Refresh***
  It enables the refresh rate feature. The chip select is released every Refresh+1 clock cycles. This parameter can be a value between 0 and 0xFFFFFFFF

55.1.2 __OSPI_HandleTypeDef

__OSPI_HandleTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***OCTOSPI_TypeDef * Instance***
- ***OSPI_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***__IO uint32_t XferSize***
- ***__IO uint32_t XferCount***
- ***DMA_HandleTypeDef * hdma***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* FifoThresholdCallback***
- ***void(* CmdCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* StatusMatchCallback***
- ***void(* TimeOutCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OCTOSPI_TypeDef* __OSPI_HandleTypeDef::Instance***
  OSPI registers base address
- ***OSPI_InitTypeDef __OSPI_HandleTypeDef::Init***
  OSPI initialization parameters
- ***uint8_t* __OSPI_HandleTypeDef::pBuffPtr***
  Address of the OSPI buffer for transfer
- ***__IO uint32_t __OSPI_HandleTypeDef::XferSize***
  Number of data to transfer
- ***__IO uint32_t __OSPI_HandleTypeDef::XferCount***
  Counter of data transferred
- ***DMA_HandleTypeDef* __OSPI_HandleTypeDef::hdma***
  Handle of the DMA channel used for the transfer

- ***__IO uint32_t __OSPI_HandleTypeDef::State***
  Internal state of the OSPI HAL driver
- ***__IO uint32_t __OSPI_HandleTypeDef::ErrorCode***
  Error code in case of HAL driver internal error
- ***uint32_t __OSPI_HandleTypeDef::Timeout***
  Timeout used for the OSPI external device access
- ***void(* __OSPI_HandleTypeDef::ErrorCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::AbortCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::FifoThresholdCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::CmdCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::RxCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::TxCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::RxHalfCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::TxHalfCpltCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::StatusMatchCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::TimeOutCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::MspInitCallback)(struct __OSPI_HandleTypeDef *hospi)***
- ***void(* __OSPI_HandleTypeDef::MspDeInitCallback)(struct __OSPI_HandleTypeDef *hospi)***

55.1.3 OSPI_RegularCmdTypeDef

OSPI_RegularCmdTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t FlashId***
- ***uint32_t Instruction***
- ***uint32_t InstructionMode***
- ***uint32_t InstructionSize***
- ***uint32_t InstructionDtrMode***
- ***uint32_t Address***
- ***uint32_t AddressMode***
- ***uint32_t AddressSize***
- ***uint32_t AddressDtrMode***
- ***uint32_t AlternateBytes***
- ***uint32_t AlternateBytesMode***
- ***uint32_t AlternateBytesSize***
- ***uint32_t AlternateBytesDtrMode***
- ***uint32_t DataMode***
- ***uint32_t NbData***
- ***uint32_t DataDtrMode***
- ***uint32_t DummyCycles***
- ***uint32_t DQSMode***
- ***uint32_t SIOOMode***

Field Documentation

- ***uint32_t OSPI_RegularCmdTypeDef::OperationType***
  It indicates if the configuration applies to the common registers or to the registers for the write operation (these registers are only used for memory-mapped mode). This parameter can be a value of ***OSPI_OperationType***
- ***uint32_t OSPI_RegularCmdTypeDef::FlashId***
  It indicates which external device is selected for this command (it applies only if Dualquad is disabled in the initialization structure). This parameter can be a value of ***OSPI_FlashID***

- ***uint32_t OSPI_RegularCmdTypeDef::Instruction***
  It contains the instruction to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionMode***
  It indicates the mode of the instruction. This parameter can be a value of ***OSPI_InstructionMode***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionSize***
  It indicates the size of the instruction. This parameter can be a value of ***OSPI_InstructionSize***
- ***uint32_t OSPI_RegularCmdTypeDef::InstructionDtrMode***
  It enables or not the DTR mode for the instruction phase. This parameter can be a value of ***OSPI_InstructionDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::AddressMode***
  It indicates the mode of the address. This parameter can be a value of ***OSPI_AddressMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressSize***
  It indicates the size of the address. This parameter can be a value of ***OSPI_AddressSize***
- ***uint32_t OSPI_RegularCmdTypeDef::AddressDtrMode***
  It enables or not the DTR mode for the address phase. This parameter can be a value of ***OSPI_AddressDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytes***
  It contains the alternate bytes to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesMode***
  It indicates the mode of the alternate bytes. This parameter can be a value of ***OSPI_AlternateBytesMode***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesSize***
  It indicates the size of the alternate bytes. This parameter can be a value of ***OSPI_AlternateBytesSize***
- ***uint32_t OSPI_RegularCmdTypeDef::AlternateBytesDtrMode***
  It enables or not the DTR mode for the alternate bytes phase. This parameter can be a value of ***OSPI_AlternateBytesDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::DataMode***
  It indicates the mode of the data. This parameter can be a value of ***OSPI_DataMode***
- ***uint32_t OSPI_RegularCmdTypeDef::NbData***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF
- ***uint32_t OSPI_RegularCmdTypeDef::DataDtrMode***
  It enables or not the DTR mode for the data phase. This parameter can be a value of ***OSPI_DataDtrMode***
- ***uint32_t OSPI_RegularCmdTypeDef::DummyCycles***
  It indicates the number of dummy cycles inserted before data phase. This parameter can be a value between 0 and 31
- ***uint32_t OSPI_RegularCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***OSPI_DQSMode***
- ***uint32_t OSPI_RegularCmdTypeDef::SIOOMode***
  It enables or not the SIOO mode. This parameter can be a value of ***OSPI_SIOOMode***

55.1.4 OSPI_HyperbusCfgTypeDef

OSPI_HyperbusCfgTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t RWRecoveryTime***
- ***uint32_t AccessTime***
- ***uint32_t WriteZeroLatency***
- ***uint32_t LatencyMode***

Field Documentation

- ***uint32_t OSPI_HyperbusCfgTypeDef::RWRecoveryTime***
  It indicates the number of cycles for the device read write recovery time. This parameter can be a value between 0 and 255
- ***uint32_t OSPI_HyperbusCfgTypeDef::AccessTime***
  It indicates the number of cycles for the device access time. This parameter can be a value between 0 and 255
- ***uint32_t OSPI_HyperbusCfgTypeDef::WriteZeroLatency***
  It enables or not the latency for the write access. This parameter can be a value of ***OSPI_WriteZeroLatency***
- ***uint32_t OSPI_HyperbusCfgTypeDef::LatencyMode***
  It configures the latency mode. This parameter can be a value of ***OSPI_LatencyMode***

55.1.5 OSPI_HyperbusCmdTypeDef

OSPI_HyperbusCmdTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t AddressSpace***
- ***uint32_t Address***
- ***uint32_t AddressSize***
- ***uint32_t NbData***
- ***uint32_t DQSMode***

Field Documentation

- ***uint32_t OSPI_HyperbusCmdTypeDef::AddressSpace***
  It indicates the address space accessed by the command. This parameter can be a value of ***OSPI_AddressSpace***
- ***uint32_t OSPI_HyperbusCmdTypeDef::Address***
  It contains the address to be sent tot he device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_HyperbusCmdTypeDef::AddressSize***
  It indicates the size of the address. This parameter can be a value of ***OSPI_AddressSize***
- ***uint32_t OSPI_HyperbusCmdTypeDef::NbData***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF In case of autopolling mode, this parameter can be any value between 1 and 4
- ***uint32_t OSPI_HyperbusCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***OSPI_DQSMode***

55.1.6 OSPI_AutoPollingTypeDef

OSPI_AutoPollingTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t Match***
- ***uint32_t Mask***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***
- ***uint32_t Interval***

Field Documentation

- ***uint32_t OSPI_AutoPollingTypeDef::Match***
  Specifies the value to be compared with the masked status register to get a match. This parameter can be any value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_AutoPollingTypeDef::Mask***
  Specifies the mask to be applied to the status bytes received. This parameter can be any value between 0 and 0xFFFFFFFF
- ***uint32_t OSPI_AutoPollingTypeDef::MatchMode***
  Specifies the method used for determining a match. This parameter can be a value of ***OSPI_MatchMode***

- ***uint32_t OSPI_AutoPollingTypeDef::AutomaticStop***
  Specifies if automatic polling is stopped after a match. This parameter can be a value of *OSPI_AutomaticStop*
- ***uint32_t OSPI_AutoPollingTypeDef::Interval***
  Specifies the number of clock cycles between two read during automatic polling phases. This parameter can be any value between 0 and 0xFFFF

55.1.7 OSPI_MemoryMappedTypeDef

OSPI_MemoryMappedTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t TimeOutActivation***
- ***uint32_t TimeOutPeriod***

Field Documentation

- ***uint32_t OSPI_MemoryMappedTypeDef::TimeOutActivation***
  Specifies if the timeout counter is enabled to release the chip select. This parameter can be a value of *OSPI_TimeOutActivation*
- ***uint32_t OSPI_MemoryMappedTypeDef::TimeOutPeriod***
  Specifies the number of clock to wait when the FIFO is full before to release the chip select. This parameter can be any value between 0 and 0xFFFF

55.1.8 OSPIM_CfgTypeDef

OSPIM_CfgTypeDef is defined in the stm32u5xx_hal_ospi.h

Data Fields

- ***uint32_t ClkPort***
- ***uint32_t DQSPort***
- ***uint32_t NCSPort***
- ***uint32_t IOLowPort***
- ***uint32_t IOHighPort***
- ***uint32_t Req2AckTime***

Field Documentation

- ***uint32_t OSPIM_CfgTypeDef::ClkPort***
  It indicates which port of the OSPI IO Manager is used for the CLK pins. This parameter can be a value between 1 and 8
- ***uint32_t OSPIM_CfgTypeDef::DQSPort***
  It indicates which port of the OSPI IO Manager is used for the DQS pin. This parameter can be a value between 0 and 8, 0 means that signal not used
- ***uint32_t OSPIM_CfgTypeDef::NCSPort***
  It indicates which port of the OSPI IO Manager is used for the NCS pin. This parameter can be a value between 1 and 8
- ***uint32_t OSPIM_CfgTypeDef::IOLowPort***
  It indicates which port of the OSPI IO Manager is used for the IO[3:0] pins. This parameter can be a value of OSPIM_IOPort
- ***uint32_t OSPIM_CfgTypeDef::IOHighPort***
  It indicates which port of the OSPI IO Manager is used for the IO[7:4] pins. This parameter can be a value of OSPIM_IOPort
- ***uint32_t OSPIM_CfgTypeDef::Req2AckTime***
  It indicates the minimum switching duration (in number of clock cycles) expected if some signals are multiplexed in the OSPI IO Manager with the other OSPI. This parameter can be a value between 1 and 256

55.2 OSPI Firmware driver API description

The following section lists the various functions of the OSPI library.

55.2.1 How to use this driver

Initialization

As prerequisite, fill in the HAL_OSPI_MspInit() :

- Enable OctoSPI and OctoSPIM clocks interface with __HAL_RCC_OSPIx_CLK_ENABLE().
- Reset OctoSPI Peripheral with __HAL_RCC_OSPIx_FORCE_RESET() and __HAL_RCC_OSPIx_RELEASE_RESET().
- Enable the clocks for the OctoSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
- Configure these OctoSPI pins in alternate mode using HAL_GPIO_Init().
- If interrupt or DMA mode is used, enable and configure OctoSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
- If DMA mode is used, enable the clocks for the OctoSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with OctoSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().

Configure the fifo threshold, the dual-quad mode, the memory type, the device size, the CS high time, the free running clock, the clock mode, the wrap size, the clock prescaler, the sample shifting, the hold delay and the CS boundary using the HAL_OSPI_Init() function.

When using Hyperbus, configure the RW recovery time, the access time, the write latency and the latency mode using the HAL_OSPI_HyperbusCfg() function.

Indirect functional mode

In regular mode, configure the command sequence using the HAL_OSPI_Command() or HAL_OSPI_Command_IT() functions :

- Instruction phase : the mode used and if present the size, the instruction opcode and the DTR mode.
- Address phase : the mode used and if present the size, the address value and the DTR mode.
- Alternate-bytes phase : the mode used and if present the size, the alternate bytes values and the DTR mode.
- Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
- Data phase : the mode used and if present the number of bytes and the DTR mode.
- Data strobe (DQS) mode : the activation (or not) of this mode
- Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
- Flash identifier : in dual-quad mode, indicates which flash is concerned
- Operation type : always common configuration

In Hyperbus mode, configure the command sequence using the HAL_OSPI_HyperbusCmd() function :

- Address space : indicate if the access will be done in register or memory
- Address size
- Number of data
- Data strobe (DQS) mode : the activation (or not) of this mode

If no data is required for the command (only for regular mode, not for Hyperbus mode), it is sent directly to the memory :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_OSPI_CmdCpltCallback() will be called when the transfer is complete.

For the indirect write mode, use HAL_OSPI_Transmit(), HAL_OSPI_Transmit_DMA() or HAL_OSPI_Transmit_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_OSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_OSPI_TxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_OSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_OSPI_TxCpltCallback() will be called when the transfer is complete.

For the indirect read mode, use HAL_OSPI_Receive(), HAL_OSPI_Receive_DMA() or HAL_OSPI_Receive_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.

- In interrupt mode, HAL_OSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_OSPI_RxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_OSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_OSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

Configure the command sequence by the same way than the indirect mode

Configure the auto-polling functional mode using the HAL_OSPI_AutoPolling() or HAL_OSPI_AutoPolling_IT() functions :

- The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.

After the configuration :

- In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
- In interrupt mode, HAL_OSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

Configure the command sequence by the same way than the indirect mode except for the operation type in regular mode :

- Operation type equals to read configuration : the command configuration applies to read access in memory-mapped mode
- Operation type equals to write configuration : the command configuration applies to write access in memory-mapped mode
- Both read and write configuration should be performed before activating memory-mapped mode

Configure the memory-mapped functional mode using the HAL_OSPI_MemoryMapped() functions :

- The timeout activation and the timeout period.

After the configuration, the OctoSPI will be used as soon as an access on the AHB is done on the address range. HAL_OSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

HAL_OSPI_GetError() function gives the error raised during the last operation.

HAL_OSPI_Abort() and HAL_OSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :

- In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
- In interrupt mode, HAL_OSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

HAL_OSPI_GetState() function gives the current state of the HAL OctoSPI driver.

HAL_OSPI_SetTimeout() function configures the timeout value used in the driver.

HAL_OSPI_SetFifoThreshold() function configures the threshold on the Fifo of the OSPI Peripheral.

HAL_OSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

IO manager configuration functions

HAL_OSPIM_Config() function configures the IO manager for the OctoSPI instance.

Callback registration

The compilation define USE_HAL_OSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use function HAL_OSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.

- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : OSPI MspInit.
- MspDeInitCallback : OSPI MspDeInit.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_OSPI_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : OSPI MspInit.
- MspDeInitCallback : OSPI MspDeInit.

This function) takes as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_OSPI_Init() and if the state is HAL_OSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_OSPI_Init() and HAL_OSPI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_OSPI_Init() and HAL_OSPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_OSPI_RegisterCallback() before calling HAL_OSPI_DeInit() or HAL_OSPI_Init() function.

When The compilation define USE_HAL_OSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

55.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the OctoSPI.
- De-initialize the OctoSPI.

This section contains the following APIs:

- *HAL_OSPI_Init()*
- *HAL_OSPI_MspInit()*
- *HAL_OSPI_DeInit()*
- *HAL_OSPI_MspDeInit()*

55.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.

- Handle the command sequence (regular and Hyperbus).
- Handle the Hyperbus configuration.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:

- *HAL_OSPI_IRQHandler()*
- *HAL_OSPI_Command()*
- *HAL_OSPI_Command_IT()*
- *HAL_OSPI_HyperbusCfg()*
- *HAL_OSPI_HyperbusCmd()*
- *HAL_OSPI_Transmit()*
- *HAL_OSPI_Receive()*
- *HAL_OSPI_Transmit_IT()*
- *HAL_OSPI_Receive_IT()*
- *HAL_OSPI_Transmit_DMA()*
- *HAL_OSPI_Receive_DMA()*
- *HAL_OSPI_AutoPolling()*
- *HAL_OSPI_AutoPolling_IT()*
- *HAL_OSPI_MemoryMapped()*
- *HAL_OSPI_ErrorCallback()*
- *HAL_OSPI_AbortCpltCallback()*
- *HAL_OSPI_FifoThresholdCallback()*
- *HAL_OSPI_CmdCpltCallback()*
- *HAL_OSPI_RxCpltCallback()*
- *HAL_OSPI_TxCpltCallback()*
- *HAL_OSPI_RxHalfCpltCallback()*
- *HAL_OSPI_TxHalfCpltCallback()*
- *HAL_OSPI_StatusMatchCallback()*
- *HAL_OSPI_TimeOutCallback()*
- *HAL_OSPI_RegisterCallback()*
- *HAL_OSPI_UnRegisterCallback()*

55.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :

- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.
- Manage the Fifo threshold.
- Configure the timeout duration used in the driver.

This section contains the following APIs:

- *HAL_OSPI_Abort()*
- *HAL_OSPI_Abort_IT()*
- *HAL_OSPI_SetFifoThreshold()*
- *HAL_OSPI_GetFifoThreshold()*
- *HAL_OSPI_SetTimeout()*
- *HAL_OSPI_GetError()*
- *HAL_OSPI_GetState()*

55.2.5 IO Manager configuration function

This subsection provides a set of functions allowing to :
- Configure the IO manager.

This section contains the following APIs:
- *HAL_OSPIM_Config()*

55.2.6 Detailed description of functions

HAL_OSPI_Init

Function name

HAL_StatusTypeDef HAL_OSPI_Init (OSPI_HandleTypeDef * hospi)

Function description

Initialize the OSPI mode according to the specified parameters in the OSPI_InitTypeDef and initialize the associated handle.

Parameters
- **hospi:** : OSPI handle

Return values
- **HAL:** status

HAL_OSPI_MspInit

Function name

void HAL_OSPI_MspInit (OSPI_HandleTypeDef * hospi)

Function description

Initialize the OSPI MSP.

Parameters
- **hospi:** : OSPI handle

Return values
- **None:**

HAL_OSPI_DeInit

Function name

HAL_StatusTypeDef HAL_OSPI_DeInit (OSPI_HandleTypeDef * hospi)

Function description

De-Initialize the OSPI peripheral.

Parameters
- **hospi:** : OSPI handle

Return values
- **HAL:** status

HAL_OSPI_MspDeInit

Function name

void HAL_OSPI_MspDeInit (OSPI_HandleTypeDef * hospi)

Function description

DeInitialize the OSPI MSP.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_IRQHandler

Function name

void HAL_OSPI_IRQHandler (OSPI_HandleTypeDef * hospi)

Function description

Handle OSPI interrupt request.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_Command

Function name

HAL_StatusTypeDef HAL_OSPI_Command (OSPI_HandleTypeDef * hospi, OSPI_RegularCmdTypeDef * cmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hospi:** : OSPI handle
- **cmd:** : structure that contains the command configuration information
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Command_IT (OSPI_HandleTypeDef * hospi, OSPI_RegularCmdTypeDef * cmd)

Function description

Set the command configuration in interrupt mode.

Parameters

- **hospi:** : OSPI handle
- **cmd:** : structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_OSPI_HyperbusCfg

Function name

HAL_StatusTypeDef HAL_OSPI_HyperbusCfg (OSPI_HandleTypeDef * hospi, OSPI_HyperbusCfgTypeDef * cfg, uint32_t Timeout)

Function description

Configure the Hyperbus parameters.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : Structure containing the Hyperbus configuration
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_HyperbusCmd

Function name

HAL_StatusTypeDef HAL_OSPI_HyperbusCmd (OSPI_HandleTypeDef * hospi, OSPI_HyperbusCmdTypeDef * cmd, uint32_t Timeout)

Function description

Set the Hyperbus command configuration.

Parameters

- **hospi:** : OSPI handle
- **cmd:** : Structure containing the Hyperbus command
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_Transmit

Function name

HAL_StatusTypeDef HAL_OSPI_Transmit (OSPI_HandleTypeDef * hospi, uint8_t * pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_OSPI_Receive

Function name

HAL_StatusTypeDef HAL_OSPI_Receive (OSPI_HandleTypeDef * hospi, uint8_t * pData, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_OSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Transmit_IT (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_OSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Receive_IT (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_OSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_OSPI_Transmit_DMA (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_OSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_OSPI_Receive_DMA (OSPI_HandleTypeDef * hospi, uint8_t * pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hospi:** : OSPI handle
- **pData:** : pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_OSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_OSPI_AutoPolling (OSPI_HandleTypeDef * hospi, OSPI_AutoPollingTypeDef * cfg, uint32_t Timeout)

Function description

Configure the OSPI Automatic Polling Mode in blocking mode.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_OSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_OSPI_AutoPolling_IT (OSPI_HandleTypeDef * hospi, OSPI_AutoPollingTypeDef * cfg)

Function description

Configure the OSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_OSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_OSPI_MemoryMapped (OSPI_HandleTypeDef * hospi, OSPI_MemoryMappedTypeDef * cfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_OSPI_ErrorCallback

Function name

void HAL_OSPI_ErrorCallback (OSPI_HandleTypeDef * hospi)

Function description

Transfer Error callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_AbortCpltCallback

Function name

void HAL_OSPI_AbortCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Abort completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_FifoThresholdCallback

Function name

void HAL_OSPI_FifoThresholdCallback (OSPI_HandleTypeDef * hospi)

Function description

FIFO Threshold callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_CmdCpltCallback

Function name

void HAL_OSPI_CmdCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Command completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_RxCpltCallback

Function name

void HAL_OSPI_RxCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Rx Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_TxCpltCallback

Function name

void HAL_OSPI_TxCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Tx Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_RxHalfCpltCallback

Function name

void HAL_OSPI_RxHalfCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_TxHalfCpltCallback

Function name

void HAL_OSPI_TxHalfCpltCallback (OSPI_HandleTypeDef * hospi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_StatusMatchCallback

Function name

void HAL_OSPI_StatusMatchCallback (OSPI_HandleTypeDef * hospi)

Function description

Status Match callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_TimeOutCallback

Function name

void HAL_OSPI_TimeOutCallback (OSPI_HandleTypeDef * hospi)

Function description

Timeout callback.

Parameters

- **hospi:** : OSPI handle

Return values

- **None:**

HAL_OSPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OSPI_RegisterCallback (OSPI_HandleTypeDef * hospi, HAL_OSPI_CallbackIDTypeDef CallbackID, pOSPI_CallbackTypeDef pCallback)

Function description

Register a User OSPI Callback To be used to override the weak predefined callback.

Parameters

- **hospi:** : OSPI handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  – HAL_OSPI_ERROR_CB_ID OSPI Error Callback ID
  – HAL_OSPI_ABORT_CB_ID OSPI Abort Callback ID
  – HAL_OSPI_FIFO_THRESHOLD_CB_ID OSPI FIFO Threshold Callback ID
  – HAL_OSPI_CMD_CPLT_CB_ID OSPI Command Complete Callback ID
  – HAL_OSPI_RX_CPLT_CB_ID OSPI Rx Complete Callback ID
  – HAL_OSPI_TX_CPLT_CB_ID OSPI Tx Complete Callback ID
  – HAL_OSPI_RX_HALF_CPLT_CB_ID OSPI Rx Half Complete Callback ID
  – HAL_OSPI_TX_HALF_CPLT_CB_ID OSPI Tx Half Complete Callback ID
  – HAL_OSPI_STATUS_MATCH_CB_ID OSPI Status Match Callback ID
  – HAL_OSPI_TIMEOUT_CB_ID OSPI Timeout Callback ID
  – HAL_OSPI_MSP_INIT_CB_ID OSPI MspInit callback ID
  – HAL_OSPI_MSP_DEINIT_CB_ID OSPI MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_OSPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OSPI_UnRegisterCallback (OSPI_HandleTypeDef * hospi, HAL_OSPI_CallbackIDTypeDef CallbackID)

Function description

Unregister a User OSPI Callback OSPI Callback is redirected to the weak predefined callback.

Parameters

- **hospi:** : OSPI handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_OSPI_ERROR_CB_ID OSPI Error Callback ID
  - HAL_OSPI_ABORT_CB_ID OSPI Abort Callback ID
  - HAL_OSPI_FIFO_THRESHOLD_CB_ID OSPI FIFO Threshold Callback ID
  - HAL_OSPI_CMD_CPLT_CB_ID OSPI Command Complete Callback ID
  - HAL_OSPI_RX_CPLT_CB_ID OSPI Rx Complete Callback ID
  - HAL_OSPI_TX_CPLT_CB_ID OSPI Tx Complete Callback ID
  - HAL_OSPI_RX_HALF_CPLT_CB_ID OSPI Rx Half Complete Callback ID
  - HAL_OSPI_TX_HALF_CPLT_CB_ID OSPI Tx Half Complete Callback ID
  - HAL_OSPI_STATUS_MATCH_CB_ID OSPI Status Match Callback ID
  - HAL_OSPI_TIMEOUT_CB_ID OSPI Timeout Callback ID
  - HAL_OSPI_MSP_INIT_CB_ID OSPI MspInit callback ID
  - HAL_OSPI_MSP_DEINIT_CB_ID OSPI MspDeInit callback ID

Return values

- **status:**

HAL_OSPI_Abort

Function name

HAL_StatusTypeDef HAL_OSPI_Abort (OSPI_HandleTypeDef * hospi)

Function description

Abort the current transmission.

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** status

HAL_OSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_OSPI_Abort_IT (OSPI_HandleTypeDef * hospi)

Function description

Abort the current transmission (non-blocking function)

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** status

HAL_OSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_OSPI_SetFifoThreshold (OSPI_HandleTypeDef * hospi, uint32_t Threshold)

Function description

Set OSPI Fifo threshold.

Parameters

- **hospi:** : OSPI handle.
- **Threshold:** : Threshold of the Fifo.

Return values

- **HAL:** status

HAL_OSPI_GetFifoThreshold

Function name

uint32_t HAL_OSPI_GetFifoThreshold (const OSPI_HandleTypeDef * hospi)

Function description

Get OSPI Fifo threshold.

Parameters

- **hospi:** : OSPI handle.

Return values

- **Fifo:** threshold

HAL_OSPI_SetTimeout

Function name

HAL_StatusTypeDef HAL_OSPI_SetTimeout (OSPI_HandleTypeDef * hospi, uint32_t Timeout)

Function description

Set OSPI timeout.

Parameters

- **hospi:** : OSPI handle.
- **Timeout:** : Timeout for the memory access.

Return values

- **None:**

HAL_OSPI_GetError

Function name

uint32_t HAL_OSPI_GetError (const OSPI_HandleTypeDef * hospi)

Function description

Return the OSPI error code.

Parameters

- **hospi:** : OSPI handle

Return values

- **OSPI:** Error Code

HAL_OSPI_GetState

Function name

uint32_t HAL_OSPI_GetState (const OSPI_HandleTypeDef * hospi)

Function description

Return the OSPI handle state.

Parameters

- **hospi:** : OSPI handle

Return values

- **HAL:** state

HAL_OSPIM_Config

Function name

HAL_StatusTypeDef HAL_OSPIM_Config (OSPI_HandleTypeDef * hospi, OSPIM_CfgTypeDef * cfg, uint32_t Timeout)

Function description

Configure the OctoSPI IO manager.

Parameters

- **hospi:** : OSPI handle
- **cfg:** : Configuration of the IO Manager for the instance
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_OSPI_DLYB_SetConfig

Function name

HAL_StatusTypeDef HAL_OSPI_DLYB_SetConfig (OSPI_HandleTypeDef * hospi, HAL_OSPI_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

HAL_OSPI_DLYB_GetConfig

Function name

HAL_StatusTypeDef HAL_OSPI_DLYB_GetConfig (const OSPI_HandleTypeDef * hospi, HAL_OSPI_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

HAL_OSPI_DLYB_GetClockPeriod

Function name

HAL_StatusTypeDef HAL_OSPI_DLYB_GetClockPeriod (OSPI_HandleTypeDef * hospi, HAL_OSPI_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

55.3 OSPI Firmware driver defines

The following section lists the various define and macros of the module.

55.3.1 OSPI

OSPI

OSPI Address DTR Mode

HAL_OSPI_ADDRESS_DTR_DISABLE

DTR mode disabled for address phase

HAL_OSPI_ADDRESS_DTR_ENABLE

DTR mode enabled for address phase

OSPI Address Mode

HAL_OSPI_ADDRESS_NONE

No address

HAL_OSPI_ADDRESS_1_LINE

Address on a single line

HAL_OSPI_ADDRESS_2_LINES

Address on two lines

HAL_OSPI_ADDRESS_4_LINES

Address on four lines

HAL_OSPI_ADDRESS_8_LINES

Address on eight lines

OSPI Address Size

HAL_OSPI_ADDRESS_8_BITS

8-bit address

HAL_OSPI_ADDRESS_16_BITS

16-bit address

HAL_OSPI_ADDRESS_24_BITS

24-bit address

HAL_OSPI_ADDRESS_32_BITS

32-bit address

OSPI Hyperbus Address Space

HAL_OSPI_MEMORY_ADDRESS_SPACE

HyperBus memory mode

HAL_OSPI_REGISTER_ADDRESS_SPACE

HyperBus register mode

OSPI Alternate Bytes DTR Mode

HAL_OSPI_ALTERNATE_BYTES_DTR_DISABLE

DTR mode disabled for alternate bytes phase

HAL_OSPI_ALTERNATE_BYTES_DTR_ENABLE

DTR mode enabled for alternate bytes phase

OSPI Alternate Bytes Mode

HAL_OSPI_ALTERNATE_BYTES_NONE

No alternate bytes

HAL_OSPI_ALTERNATE_BYTES_1_LINE

Alternate bytes on a single line

HAL_OSPI_ALTERNATE_BYTES_2_LINES
Alternate bytes on two lines

HAL_OSPI_ALTERNATE_BYTES_4_LINES
Alternate bytes on four lines

HAL_OSPI_ALTERNATE_BYTES_8_LINES
Alternate bytes on eight lines

OSPI Alternate Bytes Size

HAL_OSPI_ALTERNATE_BYTES_8_BITS
8-bit alternate bytes

HAL_OSPI_ALTERNATE_BYTES_16_BITS
16-bit alternate bytes

HAL_OSPI_ALTERNATE_BYTES_24_BITS
24-bit alternate bytes

HAL_OSPI_ALTERNATE_BYTES_32_BITS
32-bit alternate bytes

OSPI Automatic Stop

HAL_OSPI_AUTOMATIC_STOP_DISABLE
AutoPolling stops only with abort or OSPI disabling

HAL_OSPI_AUTOMATIC_STOP_ENABLE
AutoPolling stops as soon as there is a match

OSPI Clock Mode

HAL_OSPI_CLOCK_MODE_0
CLK must stay low while nCS is high

HAL_OSPI_CLOCK_MODE_3
CLK must stay high while nCS is high

OSPI Data DTR Mode

HAL_OSPI_DATA_DTR_DISABLE
DTR mode disabled for data phase

HAL_OSPI_DATA_DTR_ENABLE
DTR mode enabled for data phase

OSPI Data Mode

HAL_OSPI_DATA_NONE
No data

HAL_OSPI_DATA_1_LINE
Data on a single line

HAL_OSPI_DATA_2_LINES
Data on two lines

HAL_OSPI_DATA_4_LINES

Data on four lines

HAL_OSPI_DATA_8_LINES

Data on eight lines

OSPI Delay Block Bypaas

HAL_OSPI_DELAY_BLOCK_USED

Sampling clock is delayed by the delay block

HAL_OSPI_DELAY_BLOCK_BYPASSED

Delay block is bypassed

OSPI Delay Hold Quarter Cycle

HAL_OSPI_DHQC_DISABLE

No Delay

HAL_OSPI_DHQC_ENABLE

Delay Hold 1/4 cycle

OSPI DQS Mode

HAL_OSPI_DQS_DISABLE

DQS disabled

HAL_OSPI_DQS_ENABLE

DQS enabled

OSPI Dual-Quad

HAL_OSPI_DUALQUAD_DISABLE

Dual-Quad mode disabled

HAL_OSPI_DUALQUAD_ENABLE

Dual-Quad mode enabled

OSPI Error Code

HAL_OSPI_ERROR_NONE

No error

HAL_OSPI_ERROR_TIMEOUT

Timeout error

HAL_OSPI_ERROR_TRANSFER

Transfer error

HAL_OSPI_ERROR_DMA

DMA transfer error

HAL_OSPI_ERROR_INVALID_PARAM

Invalid parameters error

HAL_OSPI_ERROR_INVALID_SEQUENCE

Sequence of the state machine is incorrect

HAL_OSPI_ERROR_INVALID_CALLBACK

Invalid callback error

OSPI Exported Macros

__HAL_OSPI_RESET_HANDLE_STATE

Description:

- Reset OSPI handle state.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_ENABLE

Description:

- Enable the OSPI peripheral.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_DISABLE

Description:

- Disable the OSPI peripheral.

Parameters:

- __HANDLE__: specifies the OSPI Handle.

Return value:

- None

__HAL_OSPI_ENABLE_IT

Description:

- Enable the specified OSPI interrupt.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to enable. This parameter can be one of the following values:
  – HAL_OSPI_IT_TO: OSPI Timeout interrupt
  – HAL_OSPI_IT_SM: OSPI Status match interrupt
  – HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  – HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  – HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- None

__HAL_OSPI_DISABLE_IT

Description:

- Disable the specified OSPI interrupt.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to disable. This parameter can be one of the following values:
  – HAL_OSPI_IT_TO: OSPI Timeout interrupt
  – HAL_OSPI_IT_SM: OSPI Status match interrupt
  – HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  – HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  – HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- None

__HAL_OSPI_GET_IT_SOURCE

Description:

- Check whether the specified OSPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __INTERRUPT__: specifies the OSPI interrupt source to check. This parameter can be one of the following values:
  – HAL_OSPI_IT_TO: OSPI Timeout interrupt
  – HAL_OSPI_IT_SM: OSPI Status match interrupt
  – HAL_OSPI_IT_FT: OSPI FIFO threshold interrupt
  – HAL_OSPI_IT_TC: OSPI Transfer complete interrupt
  – HAL_OSPI_IT_TE: OSPI Transfer error interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_OSPI_GET_FLAG

Description:

- Check whether the selected OSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the OSPI Handle.
- __FLAG__: specifies the OSPI flag to check. This parameter can be one of the following values:
  – HAL_OSPI_FLAG_BUSY: OSPI Busy flag
  – HAL_OSPI_FLAG_TO: OSPI Timeout flag
  – HAL_OSPI_FLAG_SM: OSPI Status match flag
  – HAL_OSPI_FLAG_FT: OSPI FIFO threshold flag
  – HAL_OSPI_FLAG_TC: OSPI Transfer complete flag
  – HAL_OSPI_FLAG_TE: OSPI Transfer error flag

Return value:

- None

__HAL_OSPI_CLEAR_FLAG

Description:
- Clears the specified OSPI's flag status.

Parameters:
- __HANDLE__: specifies the OSPI Handle.
- __FLAG__: specifies the OSPI clear register flag that needs to be set This parameter can be one of the following values:
  - HAL_OSPI_FLAG_TO: OSPI Timeout flag
  - HAL_OSPI_FLAG_SM: OSPI Status match flag
  - HAL_OSPI_FLAG_TC: OSPI Transfer complete flag
  - HAL_OSPI_FLAG_TE: OSPI Transfer error flag

Return value:
- None

OSPI Exported Types

HAL_OSPI_DLYB_CfgTypeDef

OSPI Flags

HAL_OSPI_FLAG_BUSY

Busy flag: operation is ongoing

HAL_OSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

HAL_OSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

HAL_OSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

HAL_OSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

HAL_OSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

OSPI Flash Id

HAL_OSPI_FLASH_ID_1

FLASH 1 selected

HAL_OSPI_FLASH_ID_2

FLASH 2 selected

OSPI Free Running Clock

HAL_OSPI_FREERUNCLK_DISABLE

CLK is not free running

HAL_OSPI_FREERUNCLK_ENABLE

CLK is free running (always provided)

OSPI Instruction DTR Mode

HAL_OSPI_INSTRUCTION_DTR_DISABLE

DTR mode disabled for instruction phase

HAL_OSPI_INSTRUCTION_DTR_ENABLE

DTR mode enabled for instruction phase

OSPI Instruction Mode

HAL_OSPI_INSTRUCTION_NONE

No instruction

HAL_OSPI_INSTRUCTION_1_LINE

Instruction on a single line

HAL_OSPI_INSTRUCTION_2_LINES

Instruction on two lines

HAL_OSPI_INSTRUCTION_4_LINES

Instruction on four lines

HAL_OSPI_INSTRUCTION_8_LINES

Instruction on eight lines

OSPI Instruction Size

HAL_OSPI_INSTRUCTION_8_BITS

8-bit instruction

HAL_OSPI_INSTRUCTION_16_BITS

16-bit instruction

HAL_OSPI_INSTRUCTION_24_BITS

24-bit instruction

HAL_OSPI_INSTRUCTION_32_BITS

32-bit instruction

OSPI Interrupts

HAL_OSPI_IT_TO

Interrupt on the timeout flag

HAL_OSPI_IT_SM

Interrupt on the status match flag

HAL_OSPI_IT_FT

Interrupt on the fifo threshold flag

HAL_OSPI_IT_TC

Interrupt on the transfer complete flag

HAL_OSPI_IT_TE

Interrupt on the transfer error flag

OSPI IO Manager IO Port

HAL_OSPIM_IOPORT_NONE

IOs not used

HAL_OSPIM_IOPORT_1_LOW
Port 1 - IO[3:0]

HAL_OSPIM_IOPORT_1_HIGH
Port 1 - IO[7:4]

HAL_OSPIM_IOPORT_2_LOW
Port 2 - IO[3:0]

HAL_OSPIM_IOPORT_2_HIGH
Port 2 - IO[7:4]

HAL_OSPIM_IOPORT_3_LOW
Port 3 - IO[3:0]

HAL_OSPIM_IOPORT_3_HIGH
Port 3 - IO[7:4]

HAL_OSPIM_IOPORT_4_LOW
Port 4 - IO[3:0]

HAL_OSPIM_IOPORT_4_HIGH
Port 4 - IO[7:4]

HAL_OSPIM_IOPORT_5_LOW
Port 5 - IO[3:0]

HAL_OSPIM_IOPORT_5_HIGH
Port 5 - IO[7:4]

HAL_OSPIM_IOPORT_6_LOW
Port 6 - IO[3:0]

HAL_OSPIM_IOPORT_6_HIGH
Port 6 - IO[7:4]

HAL_OSPIM_IOPORT_7_LOW
Port 7 - IO[3:0]

HAL_OSPIM_IOPORT_7_HIGH
Port 7 - IO[7:4]

HAL_OSPIM_IOPORT_8_LOW
Port 8 - IO[3:0]

HAL_OSPIM_IOPORT_8_HIGH
Port 8 - IO[7:4]

OSPI Hyperbus Latency Mode

HAL_OSPI_VARIABLE_LATENCY
Variable initial latency

HAL_OSPI_FIXED_LATENCY
Fixed latency

OSPI Match Mode

HAL_OSPI_MATCH_MODE_AND

AND match mode between unmasked bits

HAL_OSPI_MATCH_MODE_OR

OR match mode between unmasked bits

OSPI Memory Type

HAL_OSPI_MEMTYPE_MICRON

Micron mode

HAL_OSPI_MEMTYPE_MACRONIX

Macronix mode

HAL_OSPI_MEMTYPE_APMEMORY

AP Memory mode

HAL_OSPI_MEMTYPE_MACRONIX_RAM

Macronix RAM mode

HAL_OSPI_MEMTYPE_HYPERBUS

Hyperbus mode

OSPI Operation Type

HAL_OSPI_OPTYPE_COMMON_CFG

Common configuration (indirect or auto-polling mode)

HAL_OSPI_OPTYPE_READ_CFG

Read configuration (memory-mapped mode)

HAL_OSPI_OPTYPE_WRITE_CFG

Write configuration (memory-mapped mode)

HAL_OSPI_OPTYPE_WRAP_CFG

Wrap configuration (memory-mapped mode)

OSPI Sample Shifting

HAL_OSPI_SAMPLE_SHIFTING_NONE

No shift

HAL_OSPI_SAMPLE_SHIFTING_HALFCYCLE

1/2 cycle shift

OSPI SIOO Mode

HAL_OSPI_SIOO_INST_EVERY_CMD

Send instruction on every transaction

HAL_OSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

OSPI State

HAL_OSPI_STATE_RESET

Initial state

HAL_OSPI_STATE_HYPERBUS_INIT

Initialization done in hyperbus mode but timing configuration not done

HAL_OSPI_STATE_READY

Driver ready to be used

HAL_OSPI_STATE_CMD_CFG

Command (regular or hyperbus) configured, ready for an action

HAL_OSPI_STATE_READ_CMD_CFG

Read command configuration done, not the write command configuration

HAL_OSPI_STATE_WRITE_CMD_CFG

Write command configuration done, not the read command configuration

HAL_OSPI_STATE_BUSY_CMD

Command without data on-going

HAL_OSPI_STATE_BUSY_TX

Indirect Tx on-going

HAL_OSPI_STATE_BUSY_RX

Indirect Rx on-going

HAL_OSPI_STATE_BUSY_AUTO_POLLING

Auto-polling on-going

HAL_OSPI_STATE_BUSY_MEM_MAPPED

Memory-mapped on-going

HAL_OSPI_STATE_ABORT

Abort on-going

HAL_OSPI_STATE_ERROR

Blocking error, driver should be re-initialized

OSPI Timeout Activation

HAL_OSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

HAL_OSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

OSPI Timeout definition

HAL_OSPI_TIMEOUT_DEFAULT_VALUE

OSPI Wrap-Size

HAL_OSPI_WRAP_NOT_SUPPORTED

wrapped reads are not supported by the memory

HAL_OSPI_WRAP_16_BYTES

external memory supports wrap size of 16 bytes

HAL_OSPI_WRAP_32_BYTES

external memory supports wrap size of 32 bytes

HAL_OSPI_WRAP_64_BYTES

external memory supports wrap size of 64 bytes

HAL_OSPI_WRAP_128_BYTES

external memory supports wrap size of 128 bytes

OSPI Hyperbus Write Zero Latency Activation

HAL_OSPI_LATENCY_ON_WRITE

Latency on write accesses

HAL_OSPI_NO_LATENCY_ON_WRITE

No latency on write accesses

56 HAL OTFDEC Generic Driver

56.1 OTFDEC Firmware driver registers structures

56.1.1 OTFDEC_RegionConfigTypeDef

OTFDEC_RegionConfigTypeDef is defined in the stm32u5xx_hal_otfdec.h

Data Fields

- ***uint32_t Nonce***
- ***uint32_t StartAddress***
- ***uint32_t EndAddress***
- ***uint16_t Version***

Field Documentation

- ***uint32_t OTFDEC_RegionConfigTypeDef::Nonce[2]***
  OTFDEC region nonce
- ***uint32_t OTFDEC_RegionConfigTypeDef::StartAddress***
  OTFDEC region start address
- ***uint32_t OTFDEC_RegionConfigTypeDef::EndAddress***
  OTFDEC region end address
- ***uint16_t OTFDEC_RegionConfigTypeDef::Version***
  OTFDEC region firmware version

56.1.2 __OTFDEC_HandleTypeDef

__OTFDEC_HandleTypeDef is defined in the stm32u5xx_hal_otfdec.h

Data Fields

- ***OTFDEC_TypeDef * Instance***
- ***HAL_OTFDEC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***OTFDEC_TypeDef* __OTFDEC_HandleTypeDef::Instance***
  OTFDEC registers base address
- ***HAL_OTFDEC_StateTypeDef __OTFDEC_HandleTypeDef::State***
  OTFDEC state
- ***HAL_LockTypeDef __OTFDEC_HandleTypeDef::Lock***
  OTFDEC locking object
- ***__IO uint32_t __OTFDEC_HandleTypeDef::ErrorCode***
  OTFDEC error code
- ***void(* __OTFDEC_HandleTypeDef::ErrorCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC error callback
- ***void(* __OTFDEC_HandleTypeDef::MspInitCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC Msp Init callback
- ***void(* __OTFDEC_HandleTypeDef::MspDeInitCallback)(struct __OTFDEC_HandleTypeDef *hotfdec)***
  OTFDEC Msp DeInit callback

56.2 OTFDEC Firmware driver API description

The following section lists the various functions of the OTFDEC library.

56.2.1 How to use this driver

The OTFDEC HAL driver can be used as follows:

1. Declare an OTFDEC_HandleTypeDef handle structure (eg. OTFDEC_HandleTypeDef hotfdec).
2. Initialize the OTFDEC low level resources by implementing the HAL_OTFDEC_MspInit() API:
   - Enable the OTFDEC interface clock.
   - NVIC configuration if interrupts are used
     - Configure the OTFDEC interrupt priority.
     - Enable the NVIC OTFDEC IRQ handle.
3. Initialize the OTFDEC peripheral by calling the HAL_OTFDEC_Init() API.
4. In the case of encryption, enable ciphering mode for the peripheral
5. For each region,
   - Configure the region deciphering mode by calling the HAL_OTFDEC_RegionSetMode() API.
   - Write the region Key by calling the HAL_OTFDEC_RegionSetKey() API. If desired, read the key CRC by calling HAL_OTFDEC_RegionGetKeyCRC() API and compare the result with the theoretically expected CRC.
   - Initialize the OTFDEC region config structure with the Nonce, protected region start and end addresses and firmware version, and wrap-up the region configuration by calling HAL_OTFDEC_RegionConfig() API.
6. At this point, the OTFDEC region configuration is done and the deciphering or enciphering enabled. The region can be deciphered on the fly after having made sure the OctoSPI is configured in memory-mapped mode or data can be enciphered by calling HAL_OTFDEC_Cipher() API.

Note: *Warning: the OTFDEC en/deciphering is based on a different endianness compared to the AES-CTR as implemented in the AES peripheral. E.g., if the OTFEC resorts to the Key (B0, B1, B2, B3) where Bi are 32-bit longwords and B0 is the Least Significant Word, the AES has to be configured with the Key (B3, B2, B1, B0) to report the same result (with the same swapping applied to the Initialization Vector).*

Callback registration

The compilation flag USE_HAL_OTFDEC_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions @ref HAL_OTFDEC_RegisterCallback() to register an interrupt callback.

Function @ref HAL_OTFDEC_RegisterCallback() allows to register following callbacks:

- ErrorCallback : OTFDEC error callback
- MspInitCallback : OTFDEC Msp Init callback
- MspDeInitCallback : OTFDEC Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function @ref HAL_OTFDEC_UnRegisterCallback to reset a callback to the default weak function.

@ref HAL_OTFDEC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- ErrorCallback : OTFDEC error callback
- MspInitCallback : OTFDEC Msp Init callback
- MspDeInitCallback : OTFDEC Msp DeInit callback

By default, after the @ref HAL_OTFDEC_Init() and when the state is @ref HAL_OTFDEC_STATE_RESET all callbacks are set to the corresponding weak functions: example @ref HAL_OTFDEC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the @ref HAL_OTFDEC_Init()/ @ref HAL_OTFDEC_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the @ref HAL_OTFDEC_Init()/ @ref HAL_OTFDEC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in @ref HAL_OTFDEC_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in @ref HAL_OTFDEC_STATE_READY or @ref HAL_OTFDEC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using @ref HAL_OTFDEC_RegisterCallback() before calling @ref HAL_OTFDEC_DeInit() or @ref HAL_OTFDEC_Init() function.

When the compilation flag USE_HAL_OTFDEC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

56.2.2 Initialization and de-initialization functions

This section contains the following APIs:
- *HAL_OTFDEC_Init()*
- *HAL_OTFDEC_DeInit()*
- *HAL_OTFDEC_MspInit()*
- *HAL_OTFDEC_MspDeInit()*
- *HAL_OTFDEC_RegisterCallback()*
- *HAL_OTFDEC_UnRegisterCallback()*

56.2.3 OTFDEC IRQ handler management

This section provides OTFDEC IRQ handler function.

This section contains the following APIs:
- *HAL_OTFDEC_IRQHandler()*
- *HAL_OTFDEC_ErrorCallback()*

56.2.4 Peripheral Control functions

This subsection permits to configure the OTFDEC peripheral

This section contains the following APIs:
- *HAL_OTFDEC_RegionKeyLock()*
- *HAL_OTFDEC_RegionSetKey()*
- *HAL_OTFDEC_RegionSetMode()*
- *HAL_OTFDEC_RegionConfig()*
- *HAL_OTFDEC_ConfigAttributes()*
- *HAL_OTFDEC_KeyCRCComputation()*
- *HAL_OTFDEC_EnableEnciphering()*
- *HAL_OTFDEC_DisableEnciphering()*
- *HAL_OTFDEC_Cipher()*
- *HAL_OTFDEC_RegionEnable()*
- *HAL_OTFDEC_RegionDisable()*

56.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:
- *HAL_OTFDEC_GetState()*
- *HAL_OTFDEC_GetConfigAttributes()*
- *HAL_OTFDEC_RegionGetKeyCRC()*
- *HAL_OTFDEC_RegionGetConfig()*

56.2.6 Detailed description of functions

HAL_OTFDEC_Init

Function name

HAL_StatusTypeDef HAL_OTFDEC_Init (OTFDEC_HandleTypeDef * hotfdec)

Function description

Initialize the OTFDEC peripheral and create the associated handle.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** status

HAL_OTFDEC_DeInit

Function name

HAL_StatusTypeDef HAL_OTFDEC_DeInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

DeInitialize the OTFDEC peripheral.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** status

HAL_OTFDEC_MspInit

Function name

void HAL_OTFDEC_MspInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

Initialize the OTFDEC MSP.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_MspDeInit

Function name

void HAL_OTFDEC_MspDeInit (OTFDEC_HandleTypeDef * hotfdec)

Function description

DeInitialize OTFDEC MSP.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegisterCallback (OTFDEC_HandleTypeDef * hotfdec, HAL_OTFDEC_CallbackIDTypeDef CallbackID, pOTFDEC_CallbackTypeDef pCallback)

Function description

Register a User OTFDEC Callback To be used instead of the weak predefined callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_OTFDEC_ERROR_CB_ID OTFDEC error callback ID
  – HAL_OTFDEC_MSPINIT_CB_ID MspInit callback ID
  – HAL_OTFDEC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_OTFDEC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_OTFDEC_UnRegisterCallback (OTFDEC_HandleTypeDef * hotfdec, HAL_OTFDEC_CallbackIDTypeDef CallbackID)

Function description

Unregister a OTFDEC Callback OTFDEC callback is redirected to the weak predefined callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_OTFDEC_ERROR_CB_ID OTFDEC error callback ID
  – HAL_OTFDEC_MSPINIT_CB_ID MspInit callback ID
  – HAL_OTFDEC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_OTFDEC_IRQHandler

Function name

void HAL_OTFDEC_IRQHandler (OTFDEC_HandleTypeDef * hotfdec)

Function description

Handle OTFDEC interrupt request.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_ErrorCallback

Function name

void HAL_OTFDEC_ErrorCallback (OTFDEC_HandleTypeDef * hotfdec)

Function description

OTFDEC error callback.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **None:**

HAL_OTFDEC_RegionKeyLock

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionKeyLock (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Lock region keys.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys of which are locked

Return values

- **HAL:** state

Notes

- Writes to this region KEYRx registers are ignored until next OTFDEC reset.

HAL_OTFDEC_RegionSetKey

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionSetKey (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, uint32_t * pKey)

Function description

Set region keys.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys of which are set
- **pKey:** pointer at set of keys

Return values

- **HAL:** state

Notes

- The API reads the key CRC computed by the peripheral and compares it with that theoretically expected. An error is reported if they are different.

HAL_OTFDEC_RegionSetMode

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionSetMode (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, uint32_t mode)

Function description

Set region mode.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the mode of which is set
- **mode:** This parameter can be only:
  - OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES All read accesses are decrypted (instruction or data)
  - OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY_WITH_CIPHER Only instruction accesses are decrypted with proprietary cipher activated

Return values

- **HAL:** state

HAL_OTFDEC_RegionConfig

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionConfig (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, const OTFDEC_RegionConfigTypeDef * Config, uint32_t lock)

Function description

Set region configuration.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region that is configured
- **Config:** pointer on structure containing the region configuration parameters
- **lock:** configuration lock enable or disable parameter This parameter can be one of the following values:
  - OTFDEC_REG_CONFIGR_LOCK_DISABLE OTFDEC region configuration is not locked
  - OTFDEC_REG_CONFIGR_LOCK_ENABLE OTFDEC region configuration is locked

Return values

- **HAL:** state

Notes

- Region enciphering/deciphering is enabled at the end of this function

HAL_OTFDEC_KeyCRCComputation

Function name

uint32_t HAL_OTFDEC_KeyCRCComputation (uint32_t * pKey)

Function description

Compute Key CRC.

Parameters

- **pKey:** pointer at set of keys

Return values

- **CRC:** value

HAL_OTFDEC_RegionEnable

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionEnable (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Enable region processing (enciphering or deciphering).

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the enciphering or deciphering is enabled

Return values

- **HAL:** state

Notes

- An error is reported when the configuration is locked.

HAL_OTFDEC_RegionDisable

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionDisable (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Disable region processing (enciphering or deciphering).

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the enciphering or deciphering is disabled

Return values

- **HAL:** state

Notes

- An error is reported when the configuration is locked.

HAL_OTFDEC_ConfigAttributes

Function name

HAL_StatusTypeDef HAL_OTFDEC_ConfigAttributes (OTFDEC_HandleTypeDef * hotfdec, uint32_t Attributes)

Function description

Configure OTFDEC attributes.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **Attributes:** This parameter can be only:
  - OTFDEC_ATTRIBUTE_PRIV Set privileged access protection
  - OTFDEC_ATTRIBUTE_NPRIV Reset privileged access protection

Return values

- **HAL:** state

Notes

- This function sets or resets regions privileged access protection.

HAL_OTFDEC_EnableEnciphering

Function name

HAL_StatusTypeDef HAL_OTFDEC_EnableEnciphering (OTFDEC_HandleTypeDef * hotfdec)

Function description

Enable peripheral enciphering.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

Notes

- By default, deciphering mode is enabled at reset

HAL_OTFDEC_DisableEnciphering

Function name

HAL_StatusTypeDef HAL_OTFDEC_DisableEnciphering (OTFDEC_HandleTypeDef * hotfdec)

Function description

Disable peripheral enciphering.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

HAL_OTFDEC_Cipher

Function name

HAL_StatusTypeDef HAL_OTFDEC_Cipher (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, const uint32_t * input, uint32_t * output, uint32_t size, uint32_t start_address)

Function description

Cipher data.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the configuration of which is used to encipher
- **input:** plain data
- **output:** ciphered data
- **size:** plain data size in words
- **start_address:** starting address in the external memory area where the enciphered data will be eventually stored

Return values

- **HAL:** state

Notes

- Region configuration parameters and OTFDEC_CR ENC bit must be set.
- output pointer points at a temporary area in RAM to store the ciphered data. It is up to the user code to copy the ciphered data in external RAM once the enciphering process is over.

HAL_OTFDEC_GetState

Function name

HAL_OTFDEC_StateTypeDef HAL_OTFDEC_GetState (OTFDEC_HandleTypeDef * hotfdec)

Function description

Return the OTFDEC state.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return values

- **HAL:** state

HAL_OTFDEC_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_OTFDEC_GetConfigAttributes (OTFDEC_HandleTypeDef * hotfdec, uint32_t * Attributes)

Function description

Get OTFDEC configuration attributes.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **Attributes:** pointer to attributes variable. This parameter can be only:
  - OTFDEC_ATTRIBUTE_PRIV Set privileged access protection
  - OTFDEC_ATTRIBUTE_NPRIV Reset privileged access protection

Return values

- **HAL:** state

Notes

- This function returns whether or not the regions access protection is in privileged mode.

HAL_OTFDEC_RegionGetKeyCRC

Function name

uint32_t HAL_OTFDEC_RegionGetKeyCRC (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex)

Function description

Return region keys CRC.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the keys CRC of which is read

Return values

- **Key:** CRC

HAL_OTFDEC_RegionGetConfig

Function name

HAL_StatusTypeDef HAL_OTFDEC_RegionGetConfig (OTFDEC_HandleTypeDef * hotfdec, uint32_t RegionIndex, OTFDEC_RegionConfigTypeDef * Config)

Function description

Return region configuration parameters.

Parameters

- **hotfdec:** pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- **RegionIndex:** index of region the configuration of which is read
- **Config:** pointer on structure that will be filled up with the region configuration parameters

Return values

- **HAL:** state

56.3 OTFDEC Firmware driver defines

The following section lists the various define and macros of the module.

56.3.1 OTFDEC

OTFDEC

OTFDEC Configuration Attributes

OTFDEC_ATTRIBUTE_NPRIV

Non-privileged access protection

OTFDEC_ATTRIBUTE_PRIV

Privileged access protection

OTFDEC Error Definition

HAL_OTFDEC_ERROR_NONE

No error

HAL_OTFDEC_SECURITY_ERROR

Security error

HAL_OTFDEC_EXECUTE_ERROR

Execute-only Execute-Never error

HAL_OTFDEC_KEY_ERROR

Key error

HAL_OTFDEC_ERROR_INVALID_CALLBACK

Invalid Callback error

OTFDEC Exported Macros

__HAL_OTFDEC_RESET_HANDLE_STATE

Description:

- Reset OTFDEC handle state.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module

Return value:

- None

__HAL_OTFDEC_ENABLE_IT

Description:

- Enable OTFDEC peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __INTERRUPT__: mask on enabled interrupts This parameter can be one of the following values:
  – OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt
  – OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt
  – OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt
  – OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts
  – OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts
  – OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts
  – OTFDEC_ALL_INT OTFDEC all interrupts

Return value:

- None

__HAL_OTFDEC_DISABLE_IT

Description:

- Disable OTFDEC peripheral interrupts combination.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __INTERRUPT__: mask on disabled interrupts This parameter can be one of the following values:
  – OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt
  – OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt
  – OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt
  – OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts
  – OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts
  – OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts
  – OTFDEC_ALL_INT OTFDEC all interrupts

Return value:

- None

__HAL_OTFDEC_GET_FLAG

Description:

- Check whether the specified combination of OTFDEC interrupt flags is set or not.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __FLAG__: mask on combination of interrupts flags This parameter can be one of the following values:
  – OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt flag
  – OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt flag
  – OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt flag
  – OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts flags
  – OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts flags
  – OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts flag
  – OTFDEC_ALL_INT OTFDEC all interrupts flags

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_OTFDEC_CLEAR_FLAG

Description:

- Clear the specified combination of OTFDEC interrupt flags.

Parameters:

- __HANDLE__: pointer to an OTFDEC_HandleTypeDef structure that contains the configuration information for OTFDEC module
- __FLAG__: mask on combination of interrupts flags This parameter can be one of the following values:
  – OTFDEC_SEC_ERROR_INT OTFDEC security error interrupt flag
  – OTFDEC_EXE_ERROR_INT OTFDEC execution error interrupt flag
  – OTFDEC_KEY_ERROR_INT OTFDEC key error interrupt flag
  – OTFDEC_SEC_EXE_ERROR_INT OTFDEC security and execution errors interrupts flags
  – OTFDEC_SEC_KEY_ERROR_INT OTFDEC security and key errors interrupts flags
  – OTFDEC_EXE_KEY_ERROR_INT OTFDEC execution and key errors interrupts flag
  – OTFDEC_ALL_INT OTFDEC all interrupts flags

Return value:

- None

OTFDEC Interrupts

OTFDEC_SEC_ERROR_INT

OTFDEC security error interrupt

OTFDEC_EXE_ERROR_INT

OTFDEC execution error interrupt

OTFDEC_KEY_ERROR_INT

OTFDEC key error interrupt

OTFDEC_SEC_EXE_ERROR_INT

OTFDEC security and execution errors interrupts

OTFDEC_SEC_KEY_ERROR_INT

OTFDEC security and key errors interrupts

OTFDEC_EXE_KEY_ERROR_INT

OTFDEC execution and key errors interrupts

OTFDEC_ALL_INT

OTFDEC all interrupts

OTFDEC Regions Index

OTFDEC_REGION1

OTFDEC region 1

OTFDEC_REGION2

OTFDEC region 2

OTFDEC_REGION3

OTFDEC region 3

OTFDEC_REGION4

OTFDEC region 4

OTFDEC Region Configuration Lock

OTFDEC_REG_CONFIGR_LOCK_DISABLE

OTFDEC region configuration lock disable

OTFDEC_REG_CONFIGR_LOCK_ENABLE

OTFDEC region configuration lock enable

OTFDEC Region Enable

OTFDEC_REG_CONFIGR_REG_DISABLE

OTFDEC region encryption or on-the-fly decryption disable

OTFDEC_REG_CONFIGR_REG_ENABLE

OTFDEC region encryption or on-the-fly decryption enable

OTFDEC Region Operating Mode

OTFDEC_REG_MODE_INSTRUCTION_OR_DATA_ACCESSES

All read accesses are decrypted

OTFDEC_REG_MODE_INSTRUCTION_ACCESSES_ONLY_WITH_CIPHER

Only instruction accesses are decrypted with proprietary cipher activated

57 HAL PCD Generic Driver

57.1 PCD Firmware driver registers structures

57.1.1 __PCD_HandleTypeDef

__PCD_HandleTypeDef is defined in the stm32u5xx_hal_pcd.h

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***
- ***PCD_EPTypeDef IN_ep***
- ***PCD_EPTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Setup***
- ***PCD_LPM_StateTypeDef LPM_State***
- ***uint32_t BESL***
- ***uint32_t FrameNumber***
- ***uint32_t lpm_active***
- ***uint32_t battery_charging_active***
- ***void * pData***
- ***void(* SOFCallback***
- ***void(* SetupStageCallback***
- ***void(* ResetCallback***
- ***void(* SuspendCallback***
- ***void(* ResumeCallback***
- ***void(* ConnectCallback***
- ***void(* DisconnectCallback***
- ***void(* DataOutStageCallback***
- ***void(* DataInStageCallback***
- ***void(* ISOOUTIncompleteCallback***
- ***void(* ISOINIncompleteCallback***
- ***void(* BCDCallback***
- ***void(* LPMCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PCD_TypeDef* __PCD_HandleTypeDef::Instance***
  Register base address
- ***PCD_InitTypeDef __PCD_HandleTypeDef::Init***
  PCD required parameters
- ***__IO uint8_t __PCD_HandleTypeDef::USB_Address***
  USB Address
- ***PCD_EPTypeDef __PCD_HandleTypeDef::IN_ep[16]***
  IN endpoint parameters
- ***PCD_EPTypeDef __PCD_HandleTypeDef::OUT_ep[16]***
  OUT endpoint parameters

- ***HAL_LockTypeDef __PCD_HandleTypeDef::Lock***
  PCD peripheral status
- ***__IO PCD_StateTypeDef __PCD_HandleTypeDef::State***
  PCD communication state
- ***__IO uint32_t __PCD_HandleTypeDef::ErrorCode***
  PCD Error code
- ***uint32_t __PCD_HandleTypeDef::Setup[12]***
  Setup packet buffer
- ***PCD_LPM_StateTypeDef __PCD_HandleTypeDef::LPM_State***
  LPM State
- ***uint32_t __PCD_HandleTypeDef::BESL***
- ***uint32_t __PCD_HandleTypeDef::FrameNumber***
  Store Current Frame number
- ***uint32_t __PCD_HandleTypeDef::lpm_active***
  Enable or disable the Link Power Management . This parameter can be set to ENABLE or DISABLE
- ***uint32_t __PCD_HandleTypeDef::battery_charging_active***
  Enable or disable Battery charging. This parameter can be set to ENABLE or DISABLE
- ***void* __PCD_HandleTypeDef::pData***
  Pointer to upper stack Handler
- ***void(* __PCD_HandleTypeDef::SOFCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD SOF callback
- ***void(* __PCD_HandleTypeDef::SetupStageCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Setup Stage callback
- ***void(* __PCD_HandleTypeDef::ResetCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Reset callback
- ***void(* __PCD_HandleTypeDef::SuspendCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Suspend callback
- ***void(* __PCD_HandleTypeDef::ResumeCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Resume callback
- ***void(* __PCD_HandleTypeDef::ConnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Connect callback
- ***void(* __PCD_HandleTypeDef::DisconnectCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Disconnect callback
- ***void(* __PCD_HandleTypeDef::DataOutStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data OUT Stage callback
- ***void(* __PCD_HandleTypeDef::DataInStageCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD Data IN Stage callback
- ***void(* __PCD_HandleTypeDef::ISOOUTIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO OUT Incomplete callback
- ***void(* __PCD_HandleTypeDef::ISOINIncompleteCallback)(struct __PCD_HandleTypeDef *hpcd, uint8_t epnum)***
  USB OTG PCD ISO IN Incomplete callback
- ***void(* __PCD_HandleTypeDef::BCDCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_BCD_MsgTypeDef msg)***
  USB OTG PCD BCD callback
- ***void(* __PCD_HandleTypeDef::LPMCallback)(struct __PCD_HandleTypeDef *hpcd, PCD_LPM_MsgTypeDef msg)***
  USB OTG PCD LPM callback
- ***void(* __PCD_HandleTypeDef::MspInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp Init callback
- ***void(* __PCD_HandleTypeDef::MspDeInitCallback)(struct __PCD_HandleTypeDef *hpcd)***
  USB OTG PCD Msp DeInit callback

57.2 PCD Firmware driver API description

The following section lists the various functions of the PCD library.

57.2.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the PCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
   a. Enable the PCD/USB Low Level interface clock using
      - __HAL_RCC_USB_CLK_ENABLE(); For USB Device FS peripheral
      - __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
      - __HAL_RCC_USB_OTG_HS_CLK_ENABLE(); (For High Speed Mode)
   b. Initialize the related GPIO clocks
   c. Configure PCD pin-out
   d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
   a. hpcd.pData = pdev;
6. Enable PCD transmission and reception:
   a. HAL_PCD_Start();

57.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- *HAL_PCD_Init()*
- *HAL_PCD_DeInit()*
- *HAL_PCD_MspInit()*
- *HAL_PCD_MspDeInit()*
- *HAL_PCD_RegisterCallback()*
- *HAL_PCD_UnRegisterCallback()*
- *HAL_PCD_RegisterDataOutStageCallback()*
- *HAL_PCD_UnRegisterDataOutStageCallback()*
- *HAL_PCD_RegisterDataInStageCallback()*
- *HAL_PCD_UnRegisterDataInStageCallback()*
- *HAL_PCD_RegisterIsoOutIncpltCallback()*
- *HAL_PCD_UnRegisterIsoOutIncpltCallback()*
- *HAL_PCD_RegisterIsoInIncpltCallback()*
- *HAL_PCD_UnRegisterIsoInIncpltCallback()*
- *HAL_PCD_RegisterBcdCallback()*
- *HAL_PCD_UnRegisterBcdCallback()*
- *HAL_PCD_RegisterLpmCallback()*
- *HAL_PCD_UnRegisterLpmCallback()*

57.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

This section contains the following APIs:

- *HAL_PCD_Start()*
- *HAL_PCD_Stop()*
- *HAL_PCD_IRQHandler()*

- *HAL_PCD_DataOutStageCallback()*
- *HAL_PCD_DataInStageCallback()*
- *HAL_PCD_SetupStageCallback()*
- *HAL_PCD_SOFCallback()*
- *HAL_PCD_ResetCallback()*
- *HAL_PCD_SuspendCallback()*
- *HAL_PCD_ResumeCallback()*
- *HAL_PCD_ISOOUTIncompleteCallback()*
- *HAL_PCD_ISOINIncompleteCallback()*
- *HAL_PCD_ConnectCallback()*
- *HAL_PCD_DisconnectCallback()*

57.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- *HAL_PCD_DevConnect()*
- *HAL_PCD_DevDisconnect()*
- *HAL_PCD_SetAddress()*
- *HAL_PCD_EP_Open()*
- *HAL_PCD_EP_Close()*
- *HAL_PCD_EP_Receive()*
- *HAL_PCD_EP_GetRxCount()*
- *HAL_PCD_EP_Transmit()*
- *HAL_PCD_EP_SetStall()*
- *HAL_PCD_EP_ClrStall()*
- *HAL_PCD_EP_Abort()*
- *HAL_PCD_EP_Flush()*
- *HAL_PCD_ActivateRemoteWakeup()*
- *HAL_PCD_DeActivateRemoteWakeup()*
- *HAL_PCD_SetTestMode()*

57.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_PCD_GetState()*
- *HAL_PCD_SetTestMode()*

57.2.6 Detailed description of functions

HAL_PCD_Init

Function name

HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and initialize the associated handle.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeInit

Function name

HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes the PCD peripheral.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_MspInit

Function name

void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)

Function description

Initializes the PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_MspDeInit

Function name

void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)

Function description

DeInitializes PCD MSP.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID, pPCD_CallbackTypeDef pCallback)

Function description

Register a User USB PCD Callback To be used instead of the weak predefined callback.

Parameters

- **hpcd:** USB PCD handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_PCD_SOF_CB_ID USB PCD SOF callback ID
  - HAL_PCD_SETUPSTAGE_CB_ID USB PCD Setup callback ID
  - HAL_PCD_RESET_CB_ID USB PCD Reset callback ID
  - HAL_PCD_SUSPEND_CB_ID USB PCD Suspend callback ID
  - HAL_PCD_RESUME_CB_ID USB PCD Resume callback ID
  - HAL_PCD_CONNECT_CB_ID USB PCD Connect callback ID
  - HAL_PCD_DISCONNECT_CB_ID USB PCD Disconnect callback ID
  - HAL_PCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_PCD_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterCallback (PCD_HandleTypeDef * hpcd, HAL_PCD_CallbackIDTypeDef CallbackID)

Function description

Unregister an USB PCD Callback USB PCD callback is redirected to the weak predefined callback.

Parameters

- **hpcd:** USB PCD handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_PCD_SOF_CB_ID USB PCD SOF callback ID
  - HAL_PCD_SETUPSTAGE_CB_ID USB PCD Setup callback ID
  - HAL_PCD_RESET_CB_ID USB PCD Reset callback ID
  - HAL_PCD_SUSPEND_CB_ID USB PCD Suspend callback ID
  - HAL_PCD_RESUME_CB_ID USB PCD Resume callback ID
  - HAL_PCD_CONNECT_CB_ID USB PCD Connect callback ID
  - HAL_PCD_DISCONNECT_CB_ID USB PCD Disconnect callback ID
  - HAL_PCD_MSPINIT_CB_ID MspDeInit callback ID
  - HAL_PCD_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_PCD_RegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataOutStageCallbackTypeDef pCallback)

Function description

Register USB PCD Data OUT Stage Callback To be used instead of the weak HAL_PCD_DataOutStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Data OUT Stage Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterDataOutStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataOutStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Data OUT Stage Callback USB PCD Data OUT Stage Callback is redirected to the weak HAL_PCD_DataOutStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterDataInStageCallback (PCD_HandleTypeDef * hpcd, pPCD_DataInStageCallbackTypeDef pCallback)

Function description

Register USB PCD Data IN Stage Callback To be used instead of the weak HAL_PCD_DataInStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Data IN Stage Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterDataInStageCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterDataInStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Data IN Stage Callback USB PCD Data OUT Stage Callback is redirected to the weak HAL_PCD_DataInStageCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoOutIncpltCallbackTypeDef pCallback)

Function description

Register USB PCD Iso OUT incomplete Callback To be used instead of the weak HAL_PCD_ISOOUTIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Iso OUT incomplete Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterIsoOutIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoOutIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Iso OUT incomplete Callback USB PCD Iso OUT incomplete Callback is redirected to the weak HAL_PCD_ISOOUTIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd, pPCD_IsoInIncpltCallbackTypeDef pCallback)

Function description

Register USB PCD Iso IN incomplete Callback To be used instead of the weak HAL_PCD_ISOINIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD Iso IN incomplete Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterIsoInIncpltCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterIsoInIncpltCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD Iso IN incomplete Callback USB PCD Iso IN incomplete Callback is redirected to the weak HAL_PCD_ISOINIncompleteCallback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterBcdCallback (PCD_HandleTypeDef * hpcd, pPCD_BcdCallbackTypeDef pCallback)

Function description

Register USB PCD BCD Callback To be used instead of the weak HAL_PCDEx_BCD_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD BCD Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterBcdCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterBcdCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD BCD Callback USB BCD Callback is redirected to the weak HAL_PCDEx_BCD_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_RegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_RegisterLpmCallback (PCD_HandleTypeDef * hpcd, pPCD_LpmCallbackTypeDef pCallback)

Function description

Register USB PCD LPM Callback To be used instead of the weak HAL_PCDEx_LPM_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle
- **pCallback:** pointer to the USB PCD LPM Callback function

Return values

- **HAL:** status

HAL_PCD_UnRegisterLpmCallback

Function name

HAL_StatusTypeDef HAL_PCD_UnRegisterLpmCallback (PCD_HandleTypeDef * hpcd)

Function description

Unregister the USB PCD LPM Callback USB LPM Callback is redirected to the weak HAL_PCDEx_LPM_Callback() predefined callback.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Start

Function name

HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)

Function description

Start the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_Stop

Function name

HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)

Function description

Stop the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_IRQHandler

Function name

void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)

Function description

Handles PCD interrupt request.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SOFCallback

Function name

void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Start Of Frame callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SetupStageCallback

Function name

void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)

Function description

Setup stage callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResetCallback

Function name

void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)

Function description

USB Reset callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_SuspendCallback

Function name

void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)

Function description

Suspend event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ResumeCallback

Function name

void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)

Function description

Resume event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_ConnectCallback

Function name

void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Connection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DisconnectCallback

Function name

void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)

Function description

Disconnection event callback.

Parameters

- **hpcd:** PCD handle

Return values

- **None:**

HAL_PCD_DataOutStageCallback

Function name

void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data OUT stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DataInStageCallback

Function name

void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Data IN stage callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOOUTIncompleteCallback

Function name

void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO OUT callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_ISOINIncompleteCallback

Function name

void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)

Function description

Incomplete ISO IN callback.

Parameters

- **hpcd:** PCD handle
- **epnum:** endpoint number

Return values

- **None:**

HAL_PCD_DevConnect

Function name

HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)

Function description

Connect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DevDisconnect

Function name

HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)

Function description

Disconnect the USB device.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetAddress

Function name

HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)

Function description

Set the USB Device address.

Parameters

- **hpcd:** PCD handle
- **address:** new device address

Return values

- **HAL:** status

HAL_PCD_EP_Open

Function name

HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)

Function description

Open and configure an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **ep_mps:** endpoint max packet size
- **ep_type:** endpoint type

Return values

- **HAL:** status

HAL_PCD_EP_Close

Function name

HAL_StatusTypeDef HAL_PCD_EP_Close (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Deactivate an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Receive

Function name

HAL_StatusTypeDef HAL_PCD_EP_Receive (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Receive an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the reception buffer
- **len:** amount of data to be received

Return values

- **HAL:** status

HAL_PCD_EP_Transmit

Function name

HAL_StatusTypeDef HAL_PCD_EP_Transmit (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf, uint32_t len)

Function description

Send an amount of data.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address
- **pBuf:** pointer to the transmission buffer
- **len:** amount of data to be sent

Return values

- **HAL:** status

HAL_PCD_EP_SetStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_SetStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Set a STALL condition over an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_ClrStall

Function name

HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Clear a STALL condition over in an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Flush

Function name

HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Flush an endpoint.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_EP_Abort

Function name

HAL_StatusTypeDef HAL_PCD_EP_Abort (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function description

Abort an USB EP transaction.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **HAL:** status

HAL_PCD_ActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

Activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_DeActivateRemoteWakeup

Function name

HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)

Function description

De-activate remote wakeup signalling.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCD_SetTestMode

Function name

HAL_StatusTypeDef HAL_PCD_SetTestMode (const PCD_HandleTypeDef * hpcd, uint8_t testmode)

Function description

Set the USB Device high speed test mode.

Parameters

- **hpcd:** PCD handle
- **testmode:** USB Device high speed test mode

Return values

- **HAL:** status

HAL_PCD_EP_GetRxCount

Function name

uint32_t HAL_PCD_EP_GetRxCount (PCD_HandleTypeDef const * hpcd, uint8_t ep_addr)

Function description

Get Received Data Size.

Parameters

- **hpcd:** PCD handle
- **ep_addr:** endpoint address

Return values

- **Data:** Size

HAL_PCD_GetState

Function name

PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef const * hpcd)

Function description

Return the PCD handle state.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** state

57.3 PCD Firmware driver defines

The following section lists the various define and macros of the module.

57.3.1 PCD

PCD

PCD Error Code definition

HAL_PCD_ERROR_INVALID_CALLBACK

Invalid Callback error

PCD Exported Macros

__HAL_PCD_ENABLE

__HAL_PCD_DISABLE

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_PCD_IS_INVALID_INTERRUPT

__HAL_PCD_UNGATE_PHYCLOCK

__HAL_PCD_GATE_PHYCLOCK

__HAL_PCD_IS_PHY_SUSPENDED

PCD PHY Module

PCD_PHY_ULPI

PCD_PHY_EMBEDDED

PCD_PHY_UTMI

PCD Speed

PCD_SPEED_HIGH

PCD_SPEED_HIGH_IN_FULL

PCD_SPEED_FULL

58 HAL PCD Extension Driver

58.1 PCDEx Firmware driver API description

The following section lists the various functions of the PCDEx library.

58.1.1 Extended features functions

This section provides functions allowing to:

- Update FIFO configuration

This section contains the following APIs:

- *HAL_PCDEx_SetTxFiFo()*
- *HAL_PCDEx_SetRxFiFo()*
- *HAL_PCDEx_ActivateLPM()*
- *HAL_PCDEx_DeActivateLPM()*
- *HAL_PCDEx_BCD_VBUSDetect()*
- *HAL_PCDEx_ActivateBCD()*
- *HAL_PCDEx_DeActivateBCD()*
- *HAL_PCDEx_LPM_Callback()*
- *HAL_PCDEx_BCD_Callback()*

58.1.2 Detailed description of functions

HAL_PCDEx_SetTxFiFo

Function name

HAL_StatusTypeDef HAL_PCDEx_SetTxFiFo (PCD_HandleTypeDef * hpcd, uint8_t fifo, uint16_t size)

Function description

Set Tx FIFO.

Parameters

- **hpcd:** PCD handle
- **fifo:** The number of Tx fifo
- **size:** Fifo size

Return values

- **HAL:** status

HAL_PCDEx_SetRxFiFo

Function name

HAL_StatusTypeDef HAL_PCDEx_SetRxFiFo (PCD_HandleTypeDef * hpcd, uint16_t size)

Function description

Set Rx FIFO.

Parameters

- **hpcd:** PCD handle
- **size:** Size of Rx fifo

Return values

- **HAL:** status

HAL_PCDEx_ActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Activate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateLPM

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateLPM (PCD_HandleTypeDef * hpcd)

Function description

Deactivate LPM feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_ActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_ActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Activate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_DeActivateBCD

Function name

HAL_StatusTypeDef HAL_PCDEx_DeActivateBCD (PCD_HandleTypeDef * hpcd)

Function description

Deactivate BatteryCharging feature.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_BCD_VBUSDetect

Function name

void HAL_PCDEx_BCD_VBUSDetect (PCD_HandleTypeDef * hpcd)

Function description

Handle BatteryCharging Process.

Parameters

- **hpcd:** PCD handle

Return values

- **HAL:** status

HAL_PCDEx_LPM_Callback

Function name

void HAL_PCDEx_LPM_Callback (PCD_HandleTypeDef * hpcd, PCD_LPM_MsgTypeDef msg)

Function description

Send LPM message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

HAL_PCDEx_BCD_Callback

Function name

void HAL_PCDEx_BCD_Callback (PCD_HandleTypeDef * hpcd, PCD_BCD_MsgTypeDef msg)

Function description

Send BatteryCharging message to user layer callback.

Parameters

- **hpcd:** PCD handle
- **msg:** LPM message

Return values

- **HAL:** status

59 HAL PKA Generic Driver

59.1 PKA Firmware driver registers structures

59.1.1 __PKA_HandleTypeDef

__PKA_HandleTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***PKA_TypeDef * Instance***
- ***__IO HAL_PKA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* OperationCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PKA_TypeDef* __PKA_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_PKA_StateTypeDef __PKA_HandleTypeDef::State***
  PKA state
- ***__IO uint32_t __PKA_HandleTypeDef::ErrorCode***
  PKA Error code
- ***void(* __PKA_HandleTypeDef::OperationCpltCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA End of operation callback
- ***void(* __PKA_HandleTypeDef::ErrorCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Error callback
- ***void(* __PKA_HandleTypeDef::MspInitCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Msp Init callback
- ***void(* __PKA_HandleTypeDef::MspDeInitCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Msp DeInit callback

59.1.2 PKA_ECCMulInTypeDef

PKA_ECCMulInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t scalarMulSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * pointX***
- ***const uint8_t * pointY***
- ***const uint8_t * scalarMul***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECCMulInTypeDef::scalarMulSize***
  Number of element in scalarMul array
- ***uint32_t PKA_ECCMulInTypeDef::modulusSize***
  Number of element in modulus, coefA, pointX and pointY arrays
- ***uint32_t PKA_ECCMulInTypeDef::coefSign***
  Curve coefficient a sign

- ***const uint8_t* PKA_ECCMulInTypeDef::coefA***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::coefB***
  pointer to curve coefficient b
- ***const uint8_t* PKA_ECCMulInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::pointX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::pointY***
  Pointer to point P coordinate yP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::scalarMul***
  Pointer to scalar multiplier k (Array of scalarMulSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::primeOrder***
  pointer to order of the curve

59.1.3 PKA_ECCMulExInTypeDef

PKA_ECCMulExInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t scalarMulSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * pointX***
- ***const uint8_t * pointY***
- ***const uint8_t * scalarMul***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECCMulExInTypeDef::primeOrderSize***
  Number of element in primeOrder array
- ***uint32_t PKA_ECCMulExInTypeDef::scalarMulSize***
  Number of element in scalarMul array
- ***uint32_t PKA_ECCMulExInTypeDef::modulusSize***
  Number of element in modulus, coefA, pointX and pointY arrays
- ***uint32_t PKA_ECCMulExInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_ECCMulExInTypeDef::coefA***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulExInTypeDef::coefB***
  pointer to curve coefficient b
- ***const uint8_t* PKA_ECCMulExInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulExInTypeDef::pointX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulExInTypeDef::pointY***
  Pointer to point P coordinate yP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulExInTypeDef::scalarMul***
  Pointer to scalar multiplier k (Array of scalarMulSize elements)
- ***const uint8_t* PKA_ECCMulExInTypeDef::primeOrder***
  pointer to order of the curve

59.1.4 PKA_PointCheckInTypeDef

PKA_PointCheckInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * pointX***
- ***const uint8_t * pointY***
- ***const uint32_t * pMontgomeryParam***

Field Documentation

- ***uint32_t PKA_PointCheckInTypeDef::modulusSize***
  Number of element in coefA, coefB, modulus, pointX and pointY arrays
- ***uint32_t PKA_PointCheckInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_PointCheckInTypeDef::coefA***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::coefB***
  Pointer to curve coefficient b (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::pointX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***const uint8_t* PKA_PointCheckInTypeDef::pointY***
  Pointer to point P coordinate yP (Array of modulusSize elements)
- ***const uint32_t* PKA_PointCheckInTypeDef::pMontgomeryParam***
  pointer to montgomery param R2 (modulus N)

59.1.5 PKA_RSACRTExpInTypeDef

PKA_RSACRTExpInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint8_t * pOpDp***
- ***const uint8_t * pOpDq***
- ***const uint8_t * pOpQinv***
- ***const uint8_t * pPrimeP***
- ***const uint8_t * pPrimeQ***
- ***const uint8_t * popA***

Field Documentation

- ***uint32_t PKA_RSACRTExpInTypeDef::size***
  Number of element in popA array
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pOpDp***
  Pointer to operand dP (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pOpDq***
  Pointer to operand dQ (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pOpQinv***
  Pointer to operand qinv (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pPrimeP***
  Pointer to prime p (Array of size/2 elements)
- ***const uint8_t* PKA_RSACRTExpInTypeDef::pPrimeQ***
  Pointer to prime Q (Array of size/2 elements)

- ***const uint8_t* PKA_RSACRTExpInTypeDef::popA***
  Pointer to operand A (Array of size elements)

59.1.6 PKA_ECDSAVerifInTypeDef

PKA_ECDSAVerifInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coef***
- ***const uint8_t * modulus***
- ***const uint8_t * basePointX***
- ***const uint8_t * basePointY***
- ***const uint8_t * pPubKeyCurvePtX***
- ***const uint8_t * pPubKeyCurvePtY***
- ***const uint8_t * RSign***
- ***const uint8_t * SSign***
- ***const uint8_t * hash***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECDSAVerifInTypeDef::primeOrderSize***
  Number of element in primeOrder array
- ***uint32_t PKA_ECDSAVerifInTypeDef::modulusSize***
  Number of element in modulus array
- ***uint32_t PKA_ECDSAVerifInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::coef***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::basePointX***
  Pointer to curve base point xG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::basePointY***
  Pointer to curve base point yG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::pPubKeyCurvePtX***
  Pointer to public-key curve point xQ (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::pPubKeyCurvePtY***
  Pointer to public-key curve point yQ (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::RSign***
  Pointer to signature part r (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::SSign***
  Pointer to signature part s (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::hash***
  Pointer to hash of the message e (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSAVerifInTypeDef::primeOrder***
  Pointer to order of the curve n (Array of primeOrderSize elements)

59.1.7 PKA_ECDSASignInTypeDef

PKA_ECDSASignInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***

- ***const uint8_t * coef***
- ***const uint8_t * coefB***
- ***const uint8_t * modulus***
- ***const uint8_t * integer***
- ***const uint8_t * basePointX***
- ***const uint8_t * basePointY***
- ***const uint8_t * hash***
- ***const uint8_t * privateKey***
- ***const uint8_t * primeOrder***

Field Documentation

- ***uint32_t PKA_ECDSASignInTypeDef::primeOrderSize***
  Number of element in primeOrder array
- ***uint32_t PKA_ECDSASignInTypeDef::modulusSize***
  Number of element in modulus array
- ***uint32_t PKA_ECDSASignInTypeDef::coefSign***
  Curve coefficient a sign
- ***const uint8_t* PKA_ECDSASignInTypeDef::coef***
  Pointer to curve coefficient |a| (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::coefB***
  Pointer to B coefficient (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::modulus***
  Pointer to curve modulus value p (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::integer***
  Pointer to random integer k (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::basePointX***
  Pointer to curve base point xG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::basePointY***
  Pointer to curve base point yG (Array of modulusSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::hash***
  Pointer to hash of the message (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::privateKey***
  Pointer to private key d (Array of primeOrderSize elements)
- ***const uint8_t* PKA_ECDSASignInTypeDef::primeOrder***
  Pointer to order of the curve n (Array of primeOrderSize elements)

59.1.8 PKA_ECDSASignOutTypeDef

PKA_ECDSASignOutTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint8_t * RSign***
- ***uint8_t * SSign***

Field Documentation

- ***uint8_t* PKA_ECDSASignOutTypeDef::RSign***
  Pointer to signature part r (Array of modulusSize elements)
- ***uint8_t* PKA_ECDSASignOutTypeDef::SSign***
  Pointer to signature part s (Array of modulusSize elements)

59.1.9 PKA_ECDSASignOutExtParamTypeDef

PKA_ECDSASignOutExtParamTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint8_t * ptX***
- ***uint8_t * ptY***

Field Documentation

- ***uint8_t* PKA_ECDSASignOutExtParamTypeDef::ptX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***uint8_t* PKA_ECDSASignOutExtParamTypeDef::ptY***
  Pointer to point P coordinate yP (Array of modulusSize elements)

59.1.10 PKA_ECCCompleteAdditionOutTypeDef

PKA_ECCCompleteAdditionOutTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint8_t * ptX***
- ***uint8_t * ptY***
- ***uint8_t * ptZ***

Field Documentation

- ***uint8_t* PKA_ECCCompleteAdditionOutTypeDef::ptX***
  pointer to point P coordinate xP
- ***uint8_t* PKA_ECCCompleteAdditionOutTypeDef::ptY***
  pointer to point P coordinate yP
- ***uint8_t* PKA_ECCCompleteAdditionOutTypeDef::ptZ***
  pointer to point P coordinate zP

59.1.11 PKA_ModExpInTypeDef

PKA_ModExpInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t expSize***
- ***uint32_t OpSize***
- ***const uint8_t * pExp***
- ***const uint8_t * pOp1***
- ***const uint8_t * pMod***

Field Documentation

- ***uint32_t PKA_ModExpInTypeDef::expSize***
  Number of element in pExp array
- ***uint32_t PKA_ModExpInTypeDef::OpSize***
  Number of element in pOp1 and pMod arrays
- ***const uint8_t* PKA_ModExpInTypeDef::pExp***
  Pointer to Exponent (Array of expSize elements)
- ***const uint8_t* PKA_ModExpInTypeDef::pOp1***
  Pointer to Operand (Array of OpSize elements)
- ***const uint8_t* PKA_ModExpInTypeDef::pMod***
  Pointer to modulus (Array of OpSize elements)

59.1.12 PKA_ModExpProtectModeInTypeDef

PKA_ModExpProtectModeInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t expSize***
- ***uint32_t OpSize***
- ***const uint8_t * pOp1***
- ***const uint8_t * pExp***
- ***const uint8_t * pMod***
- ***const uint8_t * pPhi***

Field Documentation

- ***uint32_t PKA_ModExpProtectModeInTypeDef::expSize***
  Size of the operand in bytes

- ***uint32_t PKA_ModExpProtectModeInTypeDef::OpSize***
  Size of the operand in bytes
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pOp1***
  Pointer to Operand 1
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pExp***
  Pointer to Exponent
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pMod***
  Pointer to Operand 1
- ***const uint8_t* PKA_ModExpProtectModeInTypeDef::pPhi***
  Pointer to Phi value

59.1.13 PKA_ModExpFastModeInTypeDef

PKA_ModExpFastModeInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t expSize***
- ***uint32_t OpSize***
- ***const uint8_t * pExp***
- ***const uint8_t * pOp1***
- ***const uint8_t * pMod***
- ***const uint32_t * pMontgomeryParam***

Field Documentation

- ***uint32_t PKA_ModExpFastModeInTypeDef::expSize***
  Number of element in pExp and pMontgomeryParam arrays
- ***uint32_t PKA_ModExpFastModeInTypeDef::OpSize***
  Number of element in pOp1 and pMod arrays
- ***const uint8_t* PKA_ModExpFastModeInTypeDef::pExp***
  Pointer to Exponent (Array of expSize elements)
- ***const uint8_t* PKA_ModExpFastModeInTypeDef::pOp1***
  Pointer to Operand (Array of OpSize elements)
- ***const uint8_t* PKA_ModExpFastModeInTypeDef::pMod***
  Pointer to modulus (Array of OpSize elements)
- ***const uint32_t* PKA_ModExpFastModeInTypeDef::pMontgomeryParam***
  Pointer to Montgomery parameter (Array of expSize/4 elements)

59.1.14 PKA_MontgomeryParamInTypeDef

PKA_MontgomeryParamInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint8_t * pOp1***

Field Documentation

- ***uint32_t PKA_MontgomeryParamInTypeDef::size***
  Number of element in pOp1 array
- ***const uint8_t* PKA_MontgomeryParamInTypeDef::pOp1***
  Pointer to Operand (Array of size elements)

59.1.15 PKA_AddInTypeDef

PKA_AddInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint32_t * pOp1***
- ***const uint32_t * pOp2***

Field Documentation

- ***uint32_t PKA_AddInTypeDef::size***
  Number of element in pOp1 and pOp2 arrays
- ***const uint32_t* PKA_AddInTypeDef::pOp1***
  Pointer to Operand 1 (Array of size elements)
- ***const uint32_t* PKA_AddInTypeDef::pOp2***
  Pointer to Operand 2 (Array of size elements)

59.1.16 PKA_ModInvInTypeDef

PKA_ModInvInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint32_t * pOp1***
- ***const uint8_t * pMod***

Field Documentation

- ***uint32_t PKA_ModInvInTypeDef::size***
  Number of element in pOp1 array
- ***const uint32_t* PKA_ModInvInTypeDef::pOp1***
  Pointer to Operand 1 (Array of size elements)
- ***const uint8_t* PKA_ModInvInTypeDef::pMod***
  Pointer to modulus value n (Array of size*4 elements)

59.1.17 PKA_ModRedInTypeDef

PKA_ModRedInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t OpSize***
- ***uint32_t modSize***
- ***const uint32_t * pOp1***
- ***const uint8_t * pMod***

Field Documentation

- ***uint32_t PKA_ModRedInTypeDef::OpSize***
  Number of element in pOp1 array
- ***uint32_t PKA_ModRedInTypeDef::modSize***
  Number of element in pMod array
- ***const uint32_t* PKA_ModRedInTypeDef::pOp1***
  Pointer to Operand 1 (Array of OpSize elements)
- ***const uint8_t* PKA_ModRedInTypeDef::pMod***
  Pointer to modulus value n (Array of modSize elements)

59.1.18 PKA_ModAddInTypeDef

PKA_ModAddInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t size***
- ***const uint32_t * pOp1***
- ***const uint32_t * pOp2***
- ***const uint8_t * pOp3***

Field Documentation

- ***uint32_t PKA_ModAddInTypeDef::size***
  Number of element in pOp1 and pOp2 arrays
- ***const uint32_t* PKA_ModAddInTypeDef::pOp1***
  Pointer to Operand 1 (Array of size elements)
- ***const uint32_t* PKA_ModAddInTypeDef::pOp2***
  Pointer to Operand 2 (Array of size elements)

- ***const uint8_t* PKA_ModAddInTypeDef::pOp3***
  Pointer to Operand 3 (Array of size*4 elements)

59.1.19 PKA_ECCDoubleBaseLadderInTypeDef

PKA_ECCDoubleBaseLadderInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t primeOrderSize***
- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * coefA***
- ***const uint8_t * modulus***
- ***const uint8_t * integerK***
- ***const uint8_t * integerM***
- ***const uint8_t * basePointX1***
- ***const uint8_t * basePointY1***
- ***const uint8_t * basePointZ1***
- ***const uint8_t * basePointX2***
- ***const uint8_t * basePointY2***
- ***const uint8_t * basePointZ2***

Field Documentation

- ***uint32_t PKA_ECCDoubleBaseLadderInTypeDef::primeOrderSize***
  curve prime order n length
- ***uint32_t PKA_ECCDoubleBaseLadderInTypeDef::modulusSize***
  curve modulus p length
- ***uint32_t PKA_ECCDoubleBaseLadderInTypeDef::coefSign***
  curve coefficient a sign
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::coefA***
  pointer to curve coefficient |a|
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::modulus***
  pointer to curve modulus value p
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::integerK***
  pointer to cryptographically secure random integer k
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::integerM***
  pointer to cryptographically secure random integer m
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointX1***
  pointer to curve base first point coordinate x
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointY1***
  pointer to curve base first point coordinate y
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointZ1***
  pointer to curve base first point coordinate z
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointX2***
  pointer to curve base second point coordinate x
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointY2***
  pointer to curve base second point coordinate y
- ***const uint8_t* PKA_ECCDoubleBaseLadderInTypeDef::basePointZ2***
  pointer to curve base second point coordinate z

59.1.20 PKA_ECCProjective2AffineInTypeDef

PKA_ECCProjective2AffineInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t modulusSize***
- ***const uint8_t * modulus***
- ***const uint8_t * basePointX***

- ***const uint8_t * basePointY***
- ***const uint8_t * basePointZ***
- ***const uint32_t * pMontgomeryParam***

Field Documentation

- ***uint32_t PKA_ECCProjective2AffineInTypeDef::modulusSize***
  curve modulus p length
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::modulus***
  pointer to curve modulus value p
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::basePointX***
  pointer to curve base point coordinate x
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::basePointY***
  pointer to curve base point coordinate y
- ***const uint8_t* PKA_ECCProjective2AffineInTypeDef::basePointZ***
  pointer to curve base point coordinate z
- ***const uint32_t* PKA_ECCProjective2AffineInTypeDef::pMontgomeryParam***
  pointer to montgomery parameter R2 modulus n

59.1.21 PKA_ECCCompleteAdditionInTypeDef

PKA_ECCCompleteAdditionInTypeDef is defined in the stm32u5xx_hal_pka.h

Data Fields

- ***uint32_t modulusSize***
- ***uint32_t coefSign***
- ***const uint8_t * modulus***
- ***const uint8_t * coefA***
- ***const uint8_t * basePointX1***
- ***const uint8_t * basePointY1***
- ***const uint8_t * basePointZ1***
- ***const uint8_t * basePointX2***
- ***const uint8_t * basePointY2***
- ***const uint8_t * basePointZ2***

Field Documentation

- ***uint32_t PKA_ECCCompleteAdditionInTypeDef::modulusSize***
  curve modulus p length
- ***uint32_t PKA_ECCCompleteAdditionInTypeDef::coefSign***
  curve coefficient a sign
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::modulus***
  pointer to curve modulus value p
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::coefA***
  pointer to curve coefficient |a|
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointX1***
  pointer to curve base first point coordinate x
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointY1***
  pointer to curve base first point coordinate y
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointZ1***
  pointer to curve base first point coordinate z
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointX2***
  pointer to curve base second point coordinate x
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointY2***
  pointer to curve base second point coordinate y
- ***const uint8_t* PKA_ECCCompleteAdditionInTypeDef::basePointZ2***
  pointer to curve base second point coordinate z

59.2 PKA Firmware driver API description

The following section lists the various functions of the PKA library.

59.2.1 How to use this driver

The PKA HAL driver can be used as follows:

1. Declare a PKA_HandleTypeDef handle structure, for example: PKA_HandleTypeDef hpka;
2. Initialize the PKA low level resources by implementing the HAL_PKA_MspInit() API:
   a. Enable the PKA interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the PKA interrupt priority
      - Enable the NVIC PKA IRQ Channel
3. Initialize the PKA registers by calling the HAL_PKA_Init() API which trig HAL_PKA_MspInit().
4. Fill entirely the input structure corresponding to your operation: For instance: PKA_ModExpInTypeDef for HAL_PKA_ModExp().
5. Execute the operation (in polling or interrupt) and check the returned value.
6. Retrieve the result of the operation (For instance, HAL_PKA_ModExp_GetResult for HAL_PKA_ModExp operation). The function to gather the result is different for each kind of operation. The correspondence can be found in the following section.
7. Call the function HAL_PKA_DeInit() to restore the default configuration which trig HAL_PKA_MspDeInit().

High level operation

- Input structure requires buffers as uint8_t array.
- Output structure requires buffers as uint8_t array.
- Modular exponentiation using:
  – HAL_PKA_ModExp().
  – HAL_PKA_ModExp_IT().
  – HAL_PKA_ModExpFastMode().
  – HAL_PKA_ModExpFastMode_IT().
  – HAL_PKA_ModExpProtectMode().
  – HAL_PKA_ModExpProtectMode_IT().
  – HAL_PKA_ModExp_GetResult() to retrieve the result of the operation.
- RSA Chinese Remainder Theorem (CRT) using:
  – HAL_PKA_RSACRTExp().
  – HAL_PKA_RSACRTExp_IT().
  – HAL_PKA_RSACRTExp_GetResult() to retrieve the result of the operation.
- ECC Point Check using:
  – HAL_PKA_PointCheck().
  – HAL_PKA_PointCheck_IT().
  – HAL_PKA_PointCheck_IsOnCurve() to retrieve the result of the operation.
- ECDSA Sign
  – HAL_PKA_ECDSASign().
  – HAL_PKA_ECDSASign_IT().
  – HAL_PKA_ECDSASign_GetResult() to retrieve the result of the operation.
- ECDSA Verify
  – HAL_PKA_ECDSAVerif().
  – HAL_PKA_ECDSAVerif_IT().
  – HAL_PKA_ECDSAVerif_IsValidSignature() to retrieve the result of the operation.

- ECC Scalar Multiplication using:
  – HAL_PKA_ECCMul().
  – HAL_PKA_ECCMul_IT().
  – HAL_PKA_ECCMul_GetResult() to retrieve the result of the operation.
- ECC double base ladder using:
  – HAL_PKA_ECCDoubleBaseLadder().
  – HAL_PKA_ECCDoubleBaseLadder_IT().
  – HAL_PKA_ECCDoubleBaseLadder_GetResult() to retrieve the result of the operation.
- ECC projective to affine using:
  – HAL_PKA_ECCProjective2Affine().
  – HAL_PKA_ECCProjective2Affine_IT().
  – HAL_PKA_ECCProjective2Affine_GetResult() to retrieve the result of the operation.
- ECC complete addition using:
  – HAL_PKA_ECCCompleteAddition().
  – HAL_PKA_ECCCompleteAddition_IT().
  – HAL_PKA_ECCCompleteAddition_GetResult() to retrieve the result of the operation.

Low level operation

- Input structure requires buffers as uint32_t array.
- Output structure requires buffers as uint32_t array.
- Arithmetic addition using:
  – HAL_PKA_Add().
  – HAL_PKA_Add_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation. The resulting size can be the input parameter or the input parameter size + 1 (overflow).
- Arithmetic subtraction using:
  – HAL_PKA_Sub().
  – HAL_PKA_Sub_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Arithmetic multiplication using:
  – HAL_PKA_Mul().
  – HAL_PKA_Mul_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Comparison using:
  – HAL_PKA_Cmp().
  – HAL_PKA_Cmp_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular addition using:
  – HAL_PKA_ModAdd().
  – HAL_PKA_ModAdd_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular subtraction using:
  – HAL_PKA_ModSub().
  – HAL_PKA_ModSub_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular inversion using:
  – HAL_PKA_ModInv().
  – HAL_PKA_ModInv_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.

- Modular reduction using:
  - HAL_PKA_ModRed().
  - HAL_PKA_ModRed_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Montgomery multiplication using:
  - HAL_PKA_MontgomeryMul().
  - HAL_PKA_MontgomeryMul_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.

Montgomery parameter

Polling mode operation

- When an operation is started in polling mode, the function returns when:
  - A timeout is encounter.
  - The operation is completed.

Interrupt mode operation

- Add HAL_PKA_IRQHandler to the IRQHandler of PKA.
- Enable the IRQ using HAL_NVIC_EnableIRQ().
- When an operation is started in interrupt mode, the function returns immediately.
- When the operation is completed, the callback HAL_PKA_OperationCpltCallback is called.
- When an error is encountered, the callback HAL_PKA_ErrorCallback is called.
- To stop any operation in interrupt mode, use HAL_PKA_Abort().

Utilities

- To clear the PKA RAM, use HAL_PKA_RAMReset().
- To get current state, use HAL_PKA_GetState().
- To get current error, use HAL_PKA_GetError().

Callback registration

The compilation flag USE_HAL_PKA_REGISTER_CALLBACKS, when set to 1, allows the user to configure dynamically the driver callbacks. Use Functions HAL_PKA_RegisterCallback() to register an interrupt callback.

Function HAL_PKA_RegisterCallback() allows to register following callbacks:

- OperationCpltCallback : callback for End of operation.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_PKA_UnRegisterCallback to reset a callback to the default weak function.

HAL_PKA_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- OperationCpltCallback : callback for End of operation.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_PKA_Init() and when the state is HAL_PKA_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_PKA_OperationCpltCallback(), HAL_PKA_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_PKA_Init()/ HAL_PKA_DeInit() only when these callbacks are null (not registered beforehand).

If MspInit or MspDeInit are not null, the HAL_PKA_Init()/ HAL_PKA_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_PKA_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_PKA_STATE_READY or HAL_PKA_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit.

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_PKA_RegisterCallback() before calling HAL_PKA_DeInit() or HAL_PKA_Init() function.

When the compilation flag USE_HAL_PKA_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

59.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the PKAx peripheral:

- User must implement HAL_PKA_MspInit() function in which he configures all related peripherals resources (CLOCK, IT and NVIC).
- Call the function HAL_PKA_Init() to configure the device.
- Call the function HAL_PKA_DeInit() to restore the default configuration of the selected PKAx peripheral.

This section contains the following APIs:

- ***HAL_PKA_Init()***
- ***HAL_PKA_DeInit()***
- ***HAL_PKA_MspInit()***
- ***HAL_PKA_MspDeInit()***
- ***HAL_PKA_RegisterCallback()***
- ***HAL_PKA_UnRegisterCallback()***

59.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PKA operations.

1. There are two modes of operation:
   – Blocking mode : The operation is performed in the polling mode. These functions return when data operation is completed.
   – No-Blocking mode : The operation is performed using Interrupts. These functions return immediately. The end of the operation is indicated by HAL_PKA_ErrorCallback in case of error. The end of the operation is indicated by HAL_PKA_OperationCpltCallback in case of success. To stop any operation in interrupt mode, use HAL_PKA_Abort().

2. Blocking mode functions are :
   - HAL_PKA_ModExp()
   - HAL_PKA_ModExpFastMode()
   - HAL_PKA_ModExpProtectMode()
   - HAL_PKA_ModExp_GetResult();
   - HAL_PKA_ECDSASign()
   - HAL_PKA_ECDSASign_GetResult();
   - HAL_PKA_ECDSAVerif()
   - HAL_PKA_ECDSAVerif_IsValidSignature();
   - HAL_PKA_RSACRTExp()
   - HAL_PKA_RSACRTExp_GetResult();
   - HAL_PKA_PointCheck()
   - HAL_PKA_PointCheck_IsOnCurve();
   - HAL_PKA_ECCMul()
   - HAL_PKA_ECCMulFastMode()
   - HAL_PKA_ECCMul_GetResult();
   - HAL_PKA_ECCMulEx()
   - HAL_PKA_ECCDoubleBaseLadder()
   - HAL_PKA_ECCDoubleBaseLadder_GetResult();
   - HAL_PKA_ECCProjective2Affine()
   - HAL_PKA_ECCProjective2Affine_GetResult();
   - HAL_PKA_ECCCompleteAddition()
   - HAL_PKA_ECCCompleteAddition_GetResult();
   - HAL_PKA_Add()
   - HAL_PKA_Sub()
   - HAL_PKA_Cmp()
   - HAL_PKA_Mul()
   - HAL_PKA_ModAdd()
   - HAL_PKA_ModSub()
   - HAL_PKA_ModInv()
   - HAL_PKA_ModRed()
   - HAL_PKA_MontgomeryMul()
   - HAL_PKA_Arithmetic_GetResult(P);
   - HAL_PKA_MontgomeryParam()
   - HAL_PKA_MontgomeryParam_GetResult();

3. No-Blocking mode functions with Interrupt are :
   – HAL_PKA_ModExp_IT();
   – HAL_PKA_ModExpFastMode_IT();
   – HAL_PKA_ModExpProtectMode_IT()
   – HAL_PKA_ModExp_GetResult();
   – HAL_PKA_ECDSASign_IT();
   – HAL_PKA_ECDSASign_GetResult();
   – HAL_PKA_ECDSAVerif_IT();
   – HAL_PKA_ECDSAVerif_IsValidSignature();
   – HAL_PKA_RSACRTExp_IT();
   – HAL_PKA_RSACRTExp_GetResult();
   – HAL_PKA_PointCheck_IT();
   – HAL_PKA_PointCheck_IsOnCurve();
   – HAL_PKA_ECCMul_IT();
   – HAL_PKA_ECCMulFastMode_IT();
   – HAL_PKA_ECCMul_GetResult();
   – HAL_PKA_ECCMulEx_IT();
   – HAL_PKA_ECCDoubleBaseLadder_IT()
   – HAL_PKA_ECCDoubleBaseLadder_GetResult();
   – HAL_PKA_ECCProjective2Affine_IT()
   – HAL_PKA_ECCProjective2Affine_GetResult();
   – HAL_PKA_ECCCompleteAddition_IT()
   – HAL_PKA_ECCCompleteAddition_GetResult();
   – HAL_PKA_Add_IT();
   – HAL_PKA_Sub_IT();
   – HAL_PKA_Cmp_IT();
   – HAL_PKA_Mul_IT();
   – HAL_PKA_ModAdd_IT();
   – HAL_PKA_ModSub_IT();
   – HAL_PKA_ModInv_IT();
   – HAL_PKA_ModRed_IT();
   – HAL_PKA_MontgomeryMul_IT();
   – HAL_PKA_Arithmetic_GetResult();
   – HAL_PKA_MontgomeryParam_IT();
   – HAL_PKA_MontgomeryParam_GetResult();
   – HAL_PKA_Abort();

This section contains the following APIs:
- *HAL_PKA_ModExp()*
- *HAL_PKA_ModExp_IT()*
- *HAL_PKA_ModExpFastMode()*
- *HAL_PKA_ModExpFastMode_IT()*
- *HAL_PKA_ModExpProtectMode()*
- *HAL_PKA_ModExpProtectMode_IT()*
- *HAL_PKA_ModExp_GetResult()*
- *HAL_PKA_ECDSASign()*
- *HAL_PKA_ECDSASign_IT()*
- *HAL_PKA_ECDSASign_GetResult()*
- *HAL_PKA_ECDSAVerif()*
- *HAL_PKA_ECDSAVerif_IT()*
- *HAL_PKA_ECDSAVerif_IsValidSignature()*

- *HAL_PKA_RSACRTExp()*
- *HAL_PKA_RSACRTExp_IT()*
- *HAL_PKA_RSACRTExp_GetResult()*
- *HAL_PKA_PointCheck()*
- *HAL_PKA_PointCheck_IT()*
- *HAL_PKA_PointCheck_IsOnCurve()*
- *HAL_PKA_ECCMul()*
- *HAL_PKA_ECCMul_IT()*
- *HAL_PKA_ECCMulEx()*
- *HAL_PKA_ECCMulEx_IT()*
- *HAL_PKA_ECCMul_GetResult()*
- *HAL_PKA_Add()*
- *HAL_PKA_Add_IT()*
- *HAL_PKA_Sub()*
- *HAL_PKA_Sub_IT()*
- *HAL_PKA_Mul()*
- *HAL_PKA_Mul_IT()*
- *HAL_PKA_Cmp()*
- *HAL_PKA_Cmp_IT()*
- *HAL_PKA_ModAdd()*
- *HAL_PKA_ModAdd_IT()*
- *HAL_PKA_ModInv()*
- *HAL_PKA_ModInv_IT()*
- *HAL_PKA_ModSub()*
- *HAL_PKA_ModSub_IT()*
- *HAL_PKA_ModRed()*
- *HAL_PKA_ModRed_IT()*
- *HAL_PKA_MontgomeryMul()*
- *HAL_PKA_MontgomeryMul_IT()*
- *HAL_PKA_Arithmetic_GetResult()*
- *HAL_PKA_MontgomeryParam()*
- *HAL_PKA_MontgomeryParam_IT()*
- *HAL_PKA_ECCDoubleBaseLadder()*
- *HAL_PKA_ECCDoubleBaseLadder_IT()*
- *HAL_PKA_ECCProjective2Affine()*
- *HAL_PKA_ECCProjective2Affine_IT()*
- *HAL_PKA_ECCCompleteAddition()*
- *HAL_PKA_ECCCompleteAddition_IT()*
- *HAL_PKA_MontgomeryParam_GetResult()*
- *HAL_PKA_Abort()*
- *HAL_PKA_RAMReset()*
- *HAL_PKA_IRQHandler()*
- *HAL_PKA_OperationCpltCallback()*
- *HAL_PKA_ErrorCallback()*
- *HAL_PKA_ECCDoubleBaseLadder_GetResult()*
- *HAL_PKA_ECCProjective2Affine_GetResult()*
- *HAL_PKA_ECCCompleteAddition_GetResult()*

59.2.4 Peripheral State and Error functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:
- *HAL_PKA_GetState()*
- *HAL_PKA_GetError()*

59.2.5 Detailed description of functions

HAL_PKA_Init

Function name

HAL_StatusTypeDef HAL_PKA_Init (PKA_HandleTypeDef * hpka)

Function description

Initialize the PKA according to the specified parameters in the PKA_InitTypeDef and initialize the associated handle.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_DeInit

Function name

HAL_StatusTypeDef HAL_PKA_DeInit (PKA_HandleTypeDef * hpka)

Function description

DeInitialize the PKA peripheral.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_MspInit

Function name

void HAL_PKA_MspInit (PKA_HandleTypeDef * hpka)

Function description

Initialize the PKA MSP.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_MspDeInit

Function name

void HAL_PKA_MspDeInit (PKA_HandleTypeDef * hpka)

Function description

DeInitialize the PKA MSP.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PKA_RegisterCallback (PKA_HandleTypeDef * hpka, HAL_PKA_CallbackIDTypeDef CallbackID, pPKA_CallbackTypeDef pCallback)

Function description

Register a User PKA Callback To be used instead of the weak predefined callback.

Parameters

- **hpka:** Pointer to a PKA_HandleTypeDef structure that contains the configuration information for the specified PKA.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_PKA_OPERATION_COMPLETE_CB_ID End of operation callback ID
  – HAL_PKA_ERROR_CB_ID Error callback ID
  – HAL_PKA_MSPINIT_CB_ID MspInit callback ID
  – HAL_PKA_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_PKA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PKA_UnRegisterCallback (PKA_HandleTypeDef * hpka, HAL_PKA_CallbackIDTypeDef CallbackID)

Function description

Unregister a PKA Callback PKA callback is redirected to the weak predefined callback.

Parameters

- **hpka:** Pointer to a PKA_HandleTypeDef structure that contains the configuration information for the specified PKA.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_PKA_OPERATION_COMPLETE_CB_ID End of operation callback ID
  – HAL_PKA_ERROR_CB_ID Error callback ID
  – HAL_PKA_MSPINIT_CB_ID MspInit callback ID
  – HAL_PKA_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_PKA_ModExp

Function name

HAL_StatusTypeDef HAL_PKA_ModExp (PKA_HandleTypeDef * hpka, PKA_ModExpInTypeDef * in, uint32_t Timeout)

Function description

Modular exponentiation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModExp_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModExp_IT (PKA_HandleTypeDef * hpka, PKA_ModExpInTypeDef * in)

Function description

Modular exponentiation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModExpFastMode

Function name

HAL_StatusTypeDef HAL_PKA_ModExpFastMode (PKA_HandleTypeDef * hpka, PKA_ModExpFastModeInTypeDef * in, uint32_t Timeout)

Function description

Modular exponentiation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModExpFastMode_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModExpFastMode_IT (PKA_HandleTypeDef * hpka, PKA_ModExpFastModeInTypeDef * in)

Function description

Modular exponentiation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModExpProtectMode

Function name

HAL_StatusTypeDef HAL_PKA_ModExpProtectMode (PKA_HandleTypeDef * hpka, PKA_ModExpProtectModeInTypeDef * in, uint32_t Timeout)

Function description

Modular exponentiation (protected) in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModExpProtectMode_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModExpProtectMode_IT (PKA_HandleTypeDef * hpka, PKA_ModExpProtectModeInTypeDef * in)

Function description

Modular exponentiation (protected) in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModExp_GetResult

Function name

void HAL_PKA_ModExp_GetResult (PKA_HandleTypeDef * hpka, uint8_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** Output buffer

Return values

- **HAL:** status

HAL_PKA_ECDSASign

Function name

HAL_StatusTypeDef HAL_PKA_ECDSASign (PKA_HandleTypeDef * hpka, PKA_ECDSASignInTypeDef * in, uint32_t Timeout)

Function description

Sign a message using elliptic curves over prime fields in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECDSASign_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECDSASign_IT (PKA_HandleTypeDef * hpka, PKA_ECDSASignInTypeDef * in)

Function description

Sign a message using elliptic curves over prime fields in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECDSASign_GetResult

Function name

void HAL_PKA_ECDSASign_GetResult (PKA_HandleTypeDef * hpka, PKA_ECDSASignOutTypeDef * out, PKA_ECDSASignOutExtParamTypeDef * outExt)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information
- **outExt:** Additional Output information (facultative)

HAL_PKA_ECDSAVerif

Function name

HAL_StatusTypeDef HAL_PKA_ECDSAVerif (PKA_HandleTypeDef * hpka, PKA_ECDSAVerifInTypeDef * in, uint32_t Timeout)

Function description

Verify the validity of a signature using elliptic curves over prime fields in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECDSAVerif_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECDSAVerif_IT (PKA_HandleTypeDef * hpka, PKA_ECDSAVerifInTypeDef * in)

Function description

Verify the validity of a signature using elliptic curves over prime fields in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECDSAVerif_IsValidSignature

Function name

uint32_t HAL_PKA_ECDSAVerif_IsValidSignature (PKA_HandleTypeDef const *const hpka)

Function description

Return the result of the ECDSA verification operation.

Parameters

- **hpka:** PKA handle

Return values

- **1:** if signature is verified, 0 in other case

HAL_PKA_RSACRTExp

Function name

HAL_StatusTypeDef HAL_PKA_RSACRTExp (PKA_HandleTypeDef * hpka, PKA_RSACRTExpInTypeDef * in, uint32_t Timeout)

Function description

RSA CRT exponentiation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_RSACRTExp_IT

Function name

HAL_StatusTypeDef HAL_PKA_RSACRTExp_IT (PKA_HandleTypeDef * hpka, PKA_RSACRTExpInTypeDef * in)

Function description

RSA CRT exponentiation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_RSACRTExp_GetResult

Function name

void HAL_PKA_RSACRTExp_GetResult (PKA_HandleTypeDef * hpka, uint8_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** Pointer to memory location to receive the result of the operation

Return values

- **HAL:** status

HAL_PKA_PointCheck

Function name

HAL_StatusTypeDef HAL_PKA_PointCheck (PKA_HandleTypeDef * hpka, PKA_PointCheckInTypeDef * in, uint32_t Timeout)

Function description

Point on elliptic curve check in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_PointCheck_IT

Function name

HAL_StatusTypeDef HAL_PKA_PointCheck_IT (PKA_HandleTypeDef * hpka, PKA_PointCheckInTypeDef * in)

Function description

Point on elliptic curve check in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_PointCheck_IsOnCurve

Function name

uint32_t HAL_PKA_PointCheck_IsOnCurve (PKA_HandleTypeDef const *const hpka)

Function description

Return the result of the point check operation.

Parameters

- **hpka:** PKA handle

Return values

- **1:** if point is on curve, 0 in other case

HAL_PKA_ECCMul

Function name

HAL_StatusTypeDef HAL_PKA_ECCMul (PKA_HandleTypeDef * hpka, PKA_ECCMulInTypeDef * in, uint32_t Timeout)

Function description

ECC scalar multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCMul_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCMul_IT (PKA_HandleTypeDef * hpka, PKA_ECCMulInTypeDef * in)

Function description

ECC scalar multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCMulEx

Function name

HAL_StatusTypeDef HAL_PKA_ECCMulEx (PKA_HandleTypeDef * hpka, PKA_ECCMulExInTypeDef * in, uint32_t Timeout)

Function description

ECC scalar multiplication extended in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCMulEx_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCMulEx_IT (PKA_HandleTypeDef * hpka, PKA_ECCMulExInTypeDef * in)

Function description

ECC scalar multiplication extended in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCMul_GetResult

Function name

void HAL_PKA_ECCMul_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCMulOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_Add

Function name

HAL_StatusTypeDef HAL_PKA_Add (PKA_HandleTypeDef * hpka, PKA_AddInTypeDef * in, uint32_t Timeout)

Function description

Arithmetic addition in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Add_IT

Function name

HAL_StatusTypeDef HAL_PKA_Add_IT (PKA_HandleTypeDef * hpka, PKA_AddInTypeDef * in)

Function description

Arithmetic addition in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Sub

Function name

HAL_StatusTypeDef HAL_PKA_Sub (PKA_HandleTypeDef * hpka, PKA_SubInTypeDef * in, uint32_t Timeout)

Function description

Arithmetic subtraction in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Sub_IT

Function name

HAL_StatusTypeDef HAL_PKA_Sub_IT (PKA_HandleTypeDef * hpka, PKA_SubInTypeDef * in)

Function description

Arithmetic subtraction in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Cmp

Function name

HAL_StatusTypeDef HAL_PKA_Cmp (PKA_HandleTypeDef * hpka, PKA_CmpInTypeDef * in, uint32_t Timeout)

Function description

Comparison in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Cmp_IT

Function name

HAL_StatusTypeDef HAL_PKA_Cmp_IT (PKA_HandleTypeDef * hpka, PKA_CmpInTypeDef * in)

Function description

Comparison in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Mul

Function name

HAL_StatusTypeDef HAL_PKA_Mul (PKA_HandleTypeDef * hpka, PKA_MulInTypeDef * in, uint32_t Timeout)

Function description

Arithmetic multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_Mul_IT

Function name

HAL_StatusTypeDef HAL_PKA_Mul_IT (PKA_HandleTypeDef * hpka, PKA_MulInTypeDef * in)

Function description

Arithmetic multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModAdd

Function name

HAL_StatusTypeDef HAL_PKA_ModAdd (PKA_HandleTypeDef * hpka, PKA_ModAddInTypeDef * in, uint32_t Timeout)

Function description

Modular addition in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModAdd_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModAdd_IT (PKA_HandleTypeDef * hpka, PKA_ModAddInTypeDef * in)

Function description

Modular addition in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModSub

Function name

HAL_StatusTypeDef HAL_PKA_ModSub (PKA_HandleTypeDef * hpka, PKA_ModSubInTypeDef * in, uint32_t Timeout)

Function description

Modular subtraction in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModSub_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModSub_IT (PKA_HandleTypeDef * hpka, PKA_ModSubInTypeDef * in)

Function description

Modular subtraction in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModInv

Function name

HAL_StatusTypeDef HAL_PKA_ModInv (PKA_HandleTypeDef * hpka, PKA_ModInvInTypeDef * in, uint32_t Timeout)

Function description

Modular inversion in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModInv_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModInv_IT (PKA_HandleTypeDef * hpka, PKA_ModInvInTypeDef * in)

Function description

Modular inversion in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ModRed

Function name

HAL_StatusTypeDef HAL_PKA_ModRed (PKA_HandleTypeDef * hpka, PKA_ModRedInTypeDef * in, uint32_t Timeout)

Function description

Modular reduction in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ModRed_IT

Function name

HAL_StatusTypeDef HAL_PKA_ModRed_IT (PKA_HandleTypeDef * hpka, PKA_ModRedInTypeDef * in)

Function description

Modular reduction in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_MontgomeryMul

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryMul (PKA_HandleTypeDef * hpka, PKA_MontgomeryMulInTypeDef * in, uint32_t Timeout)

Function description

Montgomery multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_MontgomeryMul_IT

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryMul_IT (PKA_HandleTypeDef * hpka, PKA_MontgomeryMulInTypeDef * in)

Function description

Montgomery multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_Arithmetic_GetResult

Function name

void HAL_PKA_Arithmetic_GetResult (PKA_HandleTypeDef * hpka, uint32_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** Pointer to memory location to receive the result of the operation

HAL_PKA_MontgomeryParam

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryParam (PKA_HandleTypeDef * hpka, PKA_MontgomeryParamInTypeDef * in, uint32_t Timeout)

Function description

Montgomery parameter computation in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_MontgomeryParam_IT

Function name

HAL_StatusTypeDef HAL_PKA_MontgomeryParam_IT (PKA_HandleTypeDef * hpka, PKA_MontgomeryParamInTypeDef * in)

Function description

Montgomery parameter computation in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_MontgomeryParam_GetResult

Function name

void HAL_PKA_MontgomeryParam_GetResult (PKA_HandleTypeDef * hpka, uint32_t * pRes)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **pRes:** pointer to buffer where the result will be copied

Return values

- **HAL:** status

HAL_PKA_ECCDoubleBaseLadder

Function name

HAL_StatusTypeDef HAL_PKA_ECCDoubleBaseLadder (PKA_HandleTypeDef * hpka, PKA_ECCDoubleBaseLadderInTypeDef * in, uint32_t Timeout)

Function description

ECC double base ladder in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCDoubleBaseLadder_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCDoubleBaseLadder_IT (PKA_HandleTypeDef * hpka, PKA_ECCDoubleBaseLadderInTypeDef * in)

Function description

ECC double base ladder in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCDoubleBaseLadder_GetResult

Function name

void HAL_PKA_ECCDoubleBaseLadder_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCDoubleBaseLadderOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_ECCProjective2Affine

Function name

HAL_StatusTypeDef HAL_PKA_ECCProjective2Affine (PKA_HandleTypeDef * hpka, PKA_ECCProjective2AffineInTypeDef * in, uint32_t Timeout)

Function description

ECC projective to affine in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCProjective2Affine_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCProjective2Affine_IT (PKA_HandleTypeDef * hpka, PKA_ECCProjective2AffineInTypeDef * in)

Function description

ECC projective to affine in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCProjective2Affine_GetResult

Function name

void HAL_PKA_ECCProjective2Affine_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCProjective2AffineOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_ECCCompleteAddition

Function name

HAL_StatusTypeDef HAL_PKA_ECCCompleteAddition (PKA_HandleTypeDef * hpka, PKA_ECCCompleteAdditionInTypeDef * in, uint32_t Timeout)

Function description

ECC complete addition in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCCompleteAddition_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCCompleteAddition_IT (PKA_HandleTypeDef * hpka, PKA_ECCCompleteAdditionInTypeDef * in)

Function description

ECC complete addition in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCCompleteAddition_GetResult

Function name

void HAL_PKA_ECCCompleteAddition_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCCompleteAdditionOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_Abort

Function name

HAL_StatusTypeDef HAL_PKA_Abort (PKA_HandleTypeDef * hpka)

Function description

Abort any ongoing operation.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_RAMReset

Function name

void HAL_PKA_RAMReset (PKA_HandleTypeDef * hpka)

Function description

Reset the PKA RAM.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_OperationCpltCallback

Function name

void HAL_PKA_OperationCpltCallback (PKA_HandleTypeDef * hpka)

Function description

Process completed callback.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_ErrorCallback

Function name

void HAL_PKA_ErrorCallback (PKA_HandleTypeDef * hpka)

Function description

Error callback.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_IRQHandler

Function name

void HAL_PKA_IRQHandler (PKA_HandleTypeDef * hpka)

Function description

This function handles PKA event interrupt request.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_GetState

Function name

HAL_PKA_StateTypeDef HAL_PKA_GetState (const PKA_HandleTypeDef * hpka)

Function description

Return the PKA handle state.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_GetError

Function name

uint32_t HAL_PKA_GetError (const PKA_HandleTypeDef * hpka)

Function description

Return the PKA error code.

Parameters

- **hpka:** PKA handle

Return values

- **PKA:** error code

59.3 PKA Firmware driver defines

The following section lists the various define and macros of the module.

59.3.1 PKA

PKA

PKA Error Code definition

HAL_PKA_ERROR_NONE

HAL_PKA_ERROR_ADDRERR

HAL_PKA_ERROR_RAMERR

HAL_PKA_ERROR_TIMEOUT

HAL_PKA_ERROR_OPERATION

HAL_PKA_ERROR_INVALID_CALLBACK

Invalid Callback error

PKA Exported Macros

__HAL_PKA_RESET_HANDLE_STATE

Description:

- Reset PKA handle state.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

__HAL_PKA_ENABLE_IT

Description:
- Enable the specified PKA interrupt.

Parameters:
- __HANDLE__: specifies the PKA Handle
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – PKA_IT_PROCEND End Of Operation interrupt enable
  – PKA_IT_ADDRERR Address error interrupt enable
  – PKA_IT_RAMERR RAM error interrupt enable
  – PKA_IT_OPERR Operation error interrupt enable

Return value:
- None

__HAL_PKA_DISABLE_IT

Description:
- Disable the specified PKA interrupt.

Parameters:
- __HANDLE__: specifies the PKA Handle
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  – PKA_IT_PROCEND End Of Operation interrupt enable
  – PKA_IT_ADDRERR Address error interrupt enable
  – PKA_IT_RAMERR RAM error interrupt enable
  – PKA_IT_OPERR Operation error interrupt enable

Return value:
- None

__HAL_PKA_GET_IT_SOURCE

Description:
- Check whether the specified PKA interrupt source is enabled or not.

Parameters:
- __HANDLE__: specifies the PKA Handle
- __INTERRUPT__: specifies the PKA interrupt source to check. This parameter can be one of the following values:
  – PKA_IT_PROCEND End Of Operation interrupt enable
  – PKA_IT_ADDRERR Address error interrupt enable
  – PKA_IT_RAMERR RAM error interrupt enable
  – PKA_IT_OPERR Operation error interrupt enable

Return value:
- The: new state of __INTERRUPT__ (SET or RESET)

__HAL_PKA_GET_FLAG

Description:

- Check whether the specified PKA flag is set or not.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - PKA_FLAG_PROCEND End Of Operation
  - PKA_FLAG_ADDRERR Address error
  - PKA_FLAG_RAMERR RAM error
  - PKA_FLAG_OPERR Operation error

Return value:

- The: new state of __FLAG__ (SET or RESET)

__HAL_PKA_CLEAR_FLAG

Description:

- Clear the PKA pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the PKA Handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - PKA_FLAG_PROCEND End Of Operation
  - PKA_FLAG_ADDRERR Address error
  - PKA_FLAG_RAMERR RAM error
  - PKA_FLAG_OPERR Operation error

Return value:

- None

__HAL_PKA_ENABLE

Description:

- Enable the specified PKA peripheral.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

__HAL_PKA_DISABLE

Description:

- Disable the specified PKA peripheral.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

__HAL_PKA_START

Description:

- Start a PKA operation.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

PKA Flag definition

PKA_FLAG_PROCEND

PKA_FLAG_ADDRERR

PKA_FLAG_RAMERR

PKA_FLAG_OPERR

PKA Interrupt configuration definition

PKA_IT_PROCEND

PKA_IT_ADDRERR

PKA_IT_RAMERR

PKA_IT_OPERR

PKA mode

PKA_MODE_MONTGOMERY_PARAM

PKA_MODE_MODULAR_EXP

PKA_MODE_MODULAR_EXP_FAST_MODE

PKA_MODE_ECC_MUL

PKA_MODE_ECDSA_SIGNATURE

PKA_MODE_ECDSA_VERIFICATION

PKA_MODE_POINT_CHECK

PKA_MODE_RSA_CRT_EXP

PKA_MODE_MODULAR_INV

PKA_MODE_ARITHMETIC_ADD

PKA_MODE_ARITHMETIC_SUB

PKA_MODE_ARITHMETIC_MUL

PKA_MODE_COMPARISON

PKA_MODE_MODULAR_RED

PKA_MODE_MODULAR_ADD

PKA_MODE_MODULAR_SUB

PKA_MODE_MONTGOMERY_MUL

PKA_MODE_ECC_PROJECTIVE_AFF

PKA_MODE_DOUBLE_BASE_LADDER

PKA_MODE_ECC_COMPLETE_ADD

PKA_MODE_MODULAR_EXP_PROTECT

PKA Operation Status

PKA_NO_ERROR

PKA_FAILED_COMPUTATION

PKA_RPART_SIGNATURE_NULL

PKA_SPART_SIGNATURE_NULL

60 HAL PSSI Generic Driver

60.1 PSSI Firmware driver registers structures

60.1.1 PSSI_InitTypeDef

PSSI_InitTypeDef is defined in the stm32u5xx_hal_pssi.h

Data Fields

- ***uint32_t DataWidth***
- ***uint32_t BusWidth***
- ***uint32_t ControlSignal***
- ***uint32_t ClockPolarity***
- ***uint32_t DataEnablePolarity***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t PSSI_InitTypeDef::DataWidth***
- ***uint32_t PSSI_InitTypeDef::BusWidth***
- ***uint32_t PSSI_InitTypeDef::ControlSignal***
- ***uint32_t PSSI_InitTypeDef::ClockPolarity***
- ***uint32_t PSSI_InitTypeDef::DataEnablePolarity***
- ***uint32_t PSSI_InitTypeDef::ReadyPolarity***

60.1.2 PSSI_HandleTypeDef

PSSI_HandleTypeDef is defined in the stm32u5xx_hal_pssi.h

Data Fields

- ***PSSI_TypeDef * Instance***
- ***PSSI_InitTypeDef Init***
- ***uint32_t * pBuffPtr***
- ***uint32_t XferCount***
- ***uint32_t XferSize***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_PSSI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***PSSI_TypeDef* PSSI_HandleTypeDef::Instance***
  PSSI register base address.
- ***PSSI_InitTypeDef PSSI_HandleTypeDef::Init***
  PSSI Initialization Structure.
- ***uint32_t* PSSI_HandleTypeDef::pBuffPtr***
  PSSI Data buffer.
- ***uint32_t PSSI_HandleTypeDef::XferCount***
  PSSI transfer count
- ***uint32_t PSSI_HandleTypeDef::XferSize***
  PSSI transfer size
- ***DMA_HandleTypeDef* PSSI_HandleTypeDef::hdmatx***
  PSSI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* PSSI_HandleTypeDef::hdmarx***
  PSSI Rx DMA Handle parameters

- ***HAL_LockTypeDef PSSI_HandleTypeDef::Lock***
  PSSI lock.
- ***__IO HAL_PSSI_StateTypeDef PSSI_HandleTypeDef::State***
  PSSI transfer state.
- ***__IO uint32_t PSSI_HandleTypeDef::ErrorCode***
  PSSI error code.

60.2 PSSI Firmware driver API description

The following section lists the various functions of the PSSI library.

60.2.1 How to use this driver

The PSSI HAL driver can be used as follows:

1. Declare a PSSI_HandleTypeDef handle structure, for example: PSSI_HandleTypeDef hpssi;
2. Initialize the PSSI low level resources by implementing the @ref HAL_PSSI_MspInit() API:
   a. Enable the PSSIx interface clock
   b. PSSI pins configuration
      - Enable the clock for the PSSI GPIOs
      - Configure PSSI pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the PSSIx interrupt priority
      - Enable the NVIC PSSI IRQ Channel
   d. DMA Configuration if you need to use DMA process
      - Declare DMA_HandleTypeDef handles structure for the transmit and receive
      - Enable the DMAx interface clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx and Rx
      - Associate the initialized DMA handle to the hpssi DMA Tx and Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx and Rx
3. Configure the Communication Bus Width, Control Signals, Input Polarity and Output Polarity in the hpssi Init structure.
4. Initialize the PSSI registers by calling the @ref HAL_PSSI_Init(), configure also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized @ref HAL_PSSI_MspInit(&hpssi) API.
5. For PSSI IO operations, two operation modes are available within this driver :

Polling mode IO operation

- Transmit an amount of data by byte in blocking mode using @ref HAL_PSSI_Transmit()
- Receive an amount of data by byte in blocking mode using @ref HAL_PSSI_Receive()

DMA mode IO operation

- Transmit an amount of data in non-blocking mode (DMA) using @ref HAL_PSSI_Transmit_DMA()
- At transmission end of transfer, @ref HAL_PSSI_TxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using @ref HAL_PSSI_Receive_DMA()
- At reception end of transfer, @ref HAL_PSSI_RxCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_RxCpltCallback()
- In case of transfer Error, @ref HAL_PSSI_ErrorCallback() function is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_ErrorCallback()
- Abort a PSSI process communication with Interrupt using @ref HAL_PSSI_Abort_IT()
- End of abort process, @ref HAL_PSSI_AbortCpltCallback() is executed and user can add his own code by customization of function pointer @ref HAL_PSSI_AbortCpltCallback()

PSSI HAL driver macros list

Below the list of most used macros in PSSI HAL driver.

- @ref HAL_PSSI_ENABLE : Enable the PSSI peripheral
- @ref HAL_PSSI_DISABLE : Disable the PSSI peripheral
- @ref HAL_PSSI_GET_FLAG : Check whether the specified PSSI flag is set or not
- @ref HAL_PSSI_CLEAR_FLAG : Clear the specified PSSI pending flag
- @ref HAL_PSSI_ENABLE_IT : Enable the specified PSSI interrupt
- @ref HAL_PSSI_DISABLE_IT : Disable the specified PSSI interrupt

Callback registration

Note: You can refer to the PSSI HAL driver header file for more useful macros

60.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the PSSIx peripheral:

- User must implement HAL_PSSI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_PSSI_Init() to configure the selected device with the selected configuration:
  - Data Width
  - Control Signals
  - Input Clock polarity
  - Output Clock polarity
- Call the function HAL_PSSI_DeInit() to restore the default configuration of the selected PSSIx peripheral.

This section contains the following APIs:

- ***HAL_PSSI_Init()***
- ***HAL_PSSI_DeInit()***
- ***HAL_PSSI_MspInit()***
- ***HAL_PSSI_MspDeInit()***

60.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PSSI data transfers.

1. There are two modes of transfer:
   - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
   - No-Blocking mode : The communication is performed using DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated the DMA IRQ .
2. Blocking mode functions are :
   - HAL_PSSI_Transmit()
   - HAL_PSSI_Receive()
3. No-Blocking mode functions with DMA are :
   - HAL_PSSI_Transmit_DMA()
   - HAL_PSSI_Receive_DMA()
4. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   - HAL_PSSI_TxCpltCallback()
   - HAL_PSSI_RxCpltCallback()
   - HAL_PSSI_ErrorCallback()
   - HAL_PSSI_AbortCpltCallback()

This section contains the following APIs:

- ***HAL_PSSI_Transmit()***

- *HAL_PSSI_Receive()*
- *HAL_PSSI_Transmit_DMA()*
- *HAL_PSSI_Receive_DMA()*
- *HAL_PSSI_Abort_DMA()*

60.2.4 Peripheral State, Mode and Error functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_PSSI_GetState()*
- *HAL_PSSI_GetError()*

60.2.5 Detailed description of functions

HAL_PSSI_Init

Function name

HAL_StatusTypeDef HAL_PSSI_Init (PSSI_HandleTypeDef * hpssi)

Function description

Initializes the PSSI according to the specified parameters in the PSSI_InitTypeDef and initialize the associated handle.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_DeInit

Function name

HAL_StatusTypeDef HAL_PSSI_DeInit (PSSI_HandleTypeDef * hpssi)

Function description

DeInitialize the PSSI peripheral.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_MspInit

Function name

void HAL_PSSI_MspInit (PSSI_HandleTypeDef * hpssi)

Function description

Initialize the PSSI MSP.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_MspDeInit

Function name

void HAL_PSSI_MspDeInit (PSSI_HandleTypeDef * hpssi)

Function description

De-Initialize the PSSI MSP.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_Transmit

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

Transmits in master mode an amount of data in blocking mode.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent (in bytes)
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PSSI_Receive

Function name

HAL_StatusTypeDef HAL_PSSI_Receive (PSSI_HandleTypeDef * hpssi, uint8_t * pData, uint32_t Size, uint32_t Timeout)

Function description

Receives an amount of data in blocking mode.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received (in bytes)
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PSSI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Transmit_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent (in bytes)

Return values

- **HAL:** status

HAL_PSSI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Receive_DMA (PSSI_HandleTypeDef * hpssi, uint32_t * pData, uint32_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received (in bytes)

Return values

- **HAL:** status

HAL_PSSI_Abort_DMA

Function name

HAL_StatusTypeDef HAL_PSSI_Abort_DMA (PSSI_HandleTypeDef * hpssi)

Function description

Abort a DMA process communication with Interrupt.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** status

HAL_PSSI_GetState

Function name

HAL_PSSI_StateTypeDef HAL_PSSI_GetState (const PSSI_HandleTypeDef * hpssi)

Function description

Return the PSSI handle state.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **HAL:** state

HAL_PSSI_GetError

Function name

uint32_t HAL_PSSI_GetError (const PSSI_HandleTypeDef * hpssi)

Function description

Return the PSSI error code.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **PSSI:** Error Code

HAL_PSSI_IRQHandler

Function name

void HAL_PSSI_IRQHandler (PSSI_HandleTypeDef * hpssi)

Function description

This function handles PSSI event interrupt request.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_TxCpltCallback

Function name

void HAL_PSSI_TxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

Tx Transfer complete callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_RxCpltCallback

Function name

void HAL_PSSI_RxCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

Rx Transfer complete callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_ErrorCallback

Function name

void HAL_PSSI_ErrorCallback (PSSI_HandleTypeDef * hpssi)

Function description

PSSI error callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

HAL_PSSI_AbortCpltCallback

Function name

void HAL_PSSI_AbortCpltCallback (PSSI_HandleTypeDef * hpssi)

Function description

PSSI abort callback.

Parameters

- **hpssi:** Pointer to a PSSI_HandleTypeDef structure that contains the configuration information for the specified PSSI.

Return values

- **None:**

60.3 PSSI Firmware driver defines

The following section lists the various define and macros of the module.

60.3.1 PSSI

PSSI

PSSI Bus Width

HAL_PSSI_8LINES

8 data lines

HAL_PSSI_16LINES

16 data lines

PSSI Data Width

HAL_PSSI_8BITS

8 Bits

HAL_PSSI_16BITS

16 Bits

HAL_PSSI_32BITS

32 Bits

PSSI definitions

PSSI_MAX_NBYTE_SIZE

PSSI_TIMEOUT_TRANSMIT

Timeout Value

PSSI_CR_OUTEN_INPUT

Input Mode

PSSI_CR_OUTEN_OUTPUT

Output Mode

PSSI_CR_DMA_ENABLE

DMA Mode Enable

PSSI_CR_DMA_DISABLE

DMA Mode Disable

PSSI_CR_16BITS

16 Lines Mode

PSSI_CR_8BITS

8 Lines Mode

PSSI_FLAG_RTT1B

1 Byte Fifo Flag

PSSI_FLAG_RTT4B

4 Bytes Fifo Flag

PSSI Error Code

HAL_PSSI_ERROR_NONE

No error

HAL_PSSI_ERROR_NOT_SUPPORTED

Not supported operation

HAL_PSSI_ERROR_UNDER_RUN

FIFO Under-run error

HAL_PSSI_ERROR_OVER_RUN

FIFO Over-run error

HAL_PSSI_ERROR_DMA

Dma error

HAL_PSSI_ERROR_TIMEOUT

Timeout error

PSSI Exported Macros

HAL_PSSI_RESET_HANDLE_STATE

Description:
- Reset PSSI handle state.

Parameters:
- __HANDLE__: specifies the PSSI handle.

Return value:
- None

HAL_PSSI_ENABLE

Description:
- Enable the PSSI.

Parameters:
- __HANDLE__: PSSI handle

Return value:
- None.

HAL_PSSI_DISABLE

Description:
- Disable the PSSI.

Parameters:
- __HANDLE__: PSSI handle

Return value:
- None.

HAL_PSSI_GET_STATUS

Description:
- Get the PSSI pending flags.

Parameters:
- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  – PSSI_FLAG_RTT1B: FIFO is ready to transfer one byte
  – PSSI_FLAG_RTT4B: FIFO is ready to transfer four bytes

Return value:
- The: state of FLAG.

HAL_PSSI_GET_FLAG

Description:
- Get the PSSI pending flags.

Parameters:
- __HANDLE__: PSSI handle
- __FLAG__: flag to check. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:
- The: state of FLAG.

HAL_PSSI_CLEAR_FLAG

Description:
- Clear the PSSI pending flags.

Parameters:
- __HANDLE__: PSSI handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Data Buffer overrun/underrun error flag

Return value:
- None

HAL_PSSI_ENABLE_IT

Description:
- Enable the specified PSSI interrupts.

Parameters:
- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be enabled. This parameter can be any combination of the following values:
  - PSSI_FLAG_OVR_RIS: Configuration error mask

Return value:
- None

HAL_PSSI_DISABLE_IT

Description:
- Disable the specified PSSI interrupts.

Parameters:
- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt sources to be disabled. This parameter can be any combination of the following values:
  - PSSI_IT_OVR_IE: Configuration error mask

Return value:
- None

HAL_PSSI_GET_IT_SOURCE

Description:
- Check whether the specified PSSI interrupt source is enabled or not.

Parameters:
- __HANDLE__: PSSI handle
- __INTERRUPT__: specifies the PSSI interrupt source to check. This parameter can be one of the following values:
  – PSSI_IT_OVR_IE: Data Buffer overrun/underrun error interrupt mask

Return value:
- The: state of INTERRUPT source.

IS_PSSI_CONTROL_SIGNAL

Description:
- Check whether the PSSI Control signal is valid.

Parameters:
- __CONTROL__: Control signals configuration

Return value:
- Valid: or not.

IS_PSSI_BUSWIDTH

Description:
- Check whether the PSSI Bus Width is valid.

Parameters:
- __BUSWIDTH__: PSSI Bush width

Return value:
- Valid: or not.

IS_PSSI_CLOCK_POLARITY

Description:
- Check whether the PSSI Clock Polarity is valid.

Parameters:
- __CLOCKPOL__: PSSI Clock Polarity

Return value:
- Valid: or not.

IS_PSSI_DE_POLARITY

Description:
- Check whether the PSSI Data Enable Polarity is valid.

Parameters:
- __DEPOL__: PSSI DE Polarity

Return value:
- Valid: or not.

IS_PSSI_RDY_POLARITY

Description:
- Check whether the PSSI Ready Polarity is valid.

Parameters:
- __RDYPOL__: PSSI RDY Polarity

Return value:
- Valid: or not.

PSSI Interrupts

PSSI_FLAG_OVR_RIS

Overrun, Underrun errors flag

PSSI_FLAG_MASK

Overrun, Underrun errors Mask

PSSI_FLAG_OVR_MIS

Overrun, Underrun masked errors flag

PSSI mode

HAL_PSSI_UNIDIRECTIONAL

Uni-directional mode

HAL_PSSI_BIDIRECTIONAL

Bi-directional mode

61 HAL PWR Generic Driver

61.1 PWR Firmware driver registers structures

61.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32u5xx_hal_pwr.h

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***
  Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_Detection_Level***.
- ***uint32_t PWR_PVDTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***.

61.2 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

61.2.1 PWR peripheral overview

1. The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller.
2. Domain architecture overview for the U5 devices: (+) U5 devices have 2 power domains (CD and SRD). The core domain (CD) contains a CPU (Cortex-M33), a Flash memory and some peripherals dedicated for general purpose. The SRD domain contains the system control and low-power peripherals.
3. Every entity has low power mode as described below :
4. The CPU low power modes are : (+) CPU CRun. (+) CPU CSleep. (+) CPU CStop.
5. The system low power modes are : (+) Run. (+) Stop 0. (+) Stop 1. (+) Stop 2. (+) Stop 3. (+) Standby. (+) Shutdown.
   ==

(#) The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller. (#) Domain architecture overview for the U5 devices:

- U5 devices have 2 power domains (CD and SRD). The core domain (CD) contains a CPU (Cortex-M33), a Flash memory and some peripherals dedicated for general purpose. The SRD domain contains the system control and low-power peripherals. (#) Every entity has low power mode as described below : (#) The CPU low power modes are :
- CPU CRun.
- CPU CSleep.
- CPU CStop. (#) The system low power modes are :
- Run.
- Stop 0.
- Stop 1.
- Stop 2.
- Stop 3.
- Standby.
- Shutdown.
  ==

61.2.2 How to use this driver

1. After startup, power management peripheral is not active by default. Use __HAL_RCC_PWR_CLK_ENABLE() macro to enable power interface.

2. Call HAL_PWR_EnableBkUpAccess() and HAL_PWR_DisableBkUpAccess() functions to enable/disable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).
3. Call HAL_PWR_ConfigPVD() after setting parameters to be configured (event mode and voltage threshold) in order to set up the Programmed Voltage Detector, then use HAL_PWR_EnablePVD() and HAL_PWR_DisablePVD() functions to start and stop the PVD detection. (+) PVD level can be one of the following values :
   – 2V0
   – 2V2
   – 2V4
   – 2V5
   – 2V6
   – 2V8
   – 2V9
   – External input analog voltage PVD_IN (compared internally to VREFINT)
4. Call HAL_PWR_EnableWakeUpPin() and HAL_PWR_DisableWakeUpPin() functions with the right parameter to configure the wake up pin polarity (Low or High), the wake up pin selection and to enable and disable it.
5. Call HAL_PWR_EnterSLEEPMode() function to enter the CPU in Sleep mode. Wake-up from Sleep mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()).
6. Call HAL_PWR_EnterSTOPMode() function to enter the whole system to Stop 0 mode. Wake-up from Stop 0 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility).
7. Call HAL_PWR_EnterSTANDBYMode() function to enter the whole system in Standby mode. Wake-up from Standby mode can be following only by an interrupt.
8. Call HAL_PWR_EnableSleepOnExit() and HAL_PWR_DisableSleepOnExit() APIs to enable and disable the Cortex-M33 re-entry in Sleep mode after an interruption handling is over.
9. Call HAL_PWR_EnableSEVOnPend() and HAL_PWR_DisableSEVOnPend() functions to configure the Cortex-M33 to wake-up after any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry.
10. Call HAL_PWR_PVD_IRQHandler() under PVD_AVD_IRQHandler() function to handle the PWR PVD interrupt request.
11. Call HAL_PWR_ConfigAttributes() function to configure PWR item secure and privilege attributes and call HAL_PWR_GetConfigAttributes() function to get the attribute configuration for the selected item.

(#) After startup, power management peripheral is not active by default. Use __HAL_RCC_PWR_CLK_ENABLE() macro to enable power interface. (#) Call HAL_PWR_EnableBkUpAccess() and HAL_PWR_DisableBkUpAccess() functions to enable/disable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM). (#) Call HAL_PWR_ConfigPVD() after setting parameters to be configured (event mode and voltage threshold) in order to set up the Programmed Voltage Detector, then use HAL_PWR_EnablePVD() and HAL_PWR_DisablePVD() functions to start and stop the PVD detection.

- PVD level can be one of the following values :
  - 2V0
  - 2V2
  - 2V4
  - 2V5
  - 2V6
  - 2V8
  - 2V9
  - External input analog voltage PVD_IN (compared internally to VREFINT) (#) Call HAL_PWR_EnableWakeUpPin() and HAL_PWR_DisableWakeUpPin() functions with the right parameter to configure the wake up pin polarity (Low or High), the wake up pin selection and to enable and disable it. (#) Call HAL_PWR_EnterSLEEPMode() function to enter the CPU in Sleep mode. Wake-up from Sleep mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (#) Call HAL_PWR_EnterSTOPMode() function to enter the whole system to Stop 0 mode. Wake-up from Stop 0 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility). (#) Call HAL_PWR_EnterSTANDBYMode() function to enter the whole system in Standby mode. Wake-up from Standby mode can be following only by an interrupt. (#) Call HAL_PWR_EnableSleepOnExit() and HAL_PWR_DisableSleepOnExit() APIs to enable and disable the Cortex-M33 re-entry in Sleep mode after an interruption handling is over. (#) Call HAL_PWR_EnableSEVOnPend() and HAL_PWR_DisableSEVOnPend() functions to configure the Cortex-M33 to wake-up after any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry. (#) Call HAL_PWR_PVD_IRQHandler() under PVD_AVD_IRQHandler() function to handle the PWR PVD interrupt request. (#) Call HAL_PWR_ConfigAttributes() function to configure PWR item secure and privilege attributes and call HAL_PWR_GetConfigAttributes() function to get the attribute configuration for the selected item.

PWR HAL driver macros list

Below the list of most used macros in PWR HAL driver.

- __HAL_PWR_GET_FLAG() : Get the PWR pending flags.
- __HAL_PWR_CLEAR_FLAG() : Clear the PWR pending flags.

61.2.3 Initialization and De-Initialization Functions

This section provides functions allowing to deinitialize power peripheral and to manage backup domain access.

After system reset, the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM) is protected against possible unwanted write accesses. The HAL_PWR_EnableBkUpAccess() function enables the access to the backup domain. The HAL_PWR_DisableBkUpAccess() function disables the access to the backup domain.

This section contains the following APIs:

- *HAL_PWR_DeInit()*
- *HAL_PWR_EnableBkUpAccess()*
- *HAL_PWR_DisableBkUpAccess()*

61.2.4 Peripheral Control functions

This section provides functions allowing to control power peripheral.

PVD configuration

- The PVD can be used to monitor the VDD power supply by comparing it to a threshold selected by the PVDLS[2:0] bits in the PWR supply voltage monitoring control register (PWR_SVMCR) and can be enabled by setting the PVDE bit.
- A PVDO flag is available in the PWR supply voltage monitoring control register (PWR_SVMCR) to indicate if VDD is higher or lower than the PVD threshold. This event is internally connected to the EXTI line 16 and can generate an interrupt if enabled through the EXTI registers. It is configurable through __HAL_PWR_PVD_EXTI_ENABLE_IT() macro.

- The PVD can remain active in Stop 0, Stop 1, Stop 2 modes, and the PVM interrupt can wake up from the Stop mode. The PVD is not functional in Stop 3 mode.
- During Stop 1, Stop 2 and Stop 3 modes, it is possible to set the PVD in ultra-low-power mode to further reduce the current consumption by setting the ULPMEN bit in PWR_CR1 register.

Wake-up pin configuration

- Wake-up pin is used to wake up the system from Stop 3, Standby and Shutdown mode. The pin selection is configurable through the WUCR3 register to map internal signal to wake up pin line. The pin polarity is configurable through the WUCR2 register to be active on rising or falling edges.
- There are up to 24 wake-up signals that can be mapped to up to 8 wake-up lines in the STM32U5 family.

Low Power modes configuration

This section presents 3 principles low-power modes :

- Sleep mode : Cortex-M33 is stopped and all PWR domains are remaining active (powered and clocked).
- Stop 0 mode : Cortex-M33 is stopped, clocks are stopped and the regulator is running.
- Standby mode : All PWR domains enter DSTANDBY mode and the VCORE supply regulator is powered off.

Sleep mode

- Entry : The Sleep mode is entered by using the HAL_PWR_EnterSLEEPMode() function.
  - PWR_SLEEPENTRY_WFI : Enter SLEEP mode with WFI instruction.
  - PWR_SLEEPENTRY_WFE : Enter SLEEP mode with WFE instruction and clear of pending events before.
  - PWR_SLEEPENTRY_WFE_NO_EVT_CLEAR: Enter SLEEP mode with WFE instruction and no clear of pending event before.

Note: The Regulator parameter is not used for the STM32U5 family and is kept as parameter just to maintain compatibility with other families.

- Exit : According to Sleep entry, any event when entry is __WFE() intrinsic and any interrupt when entry is __WFI() intrinsic can wake up the device from Sleep mode.

Stop 0 mode

The Stop 0 mode is based on the Cortex-M33 Deepsleep mode combined with the peripheral clock gating. The voltage regulator is configured in main regulator mode. In Stop 0 mode, all clocks in the VCORE domain are stopped. The PLL, MSIS, MSIK, HSI16 and HSE oscillators are disabled. Some peripherals with the LPBAM capability can switch on HSI16 or MSIS or MSIK for transferring data. All SRAMs and register contents are preserved, but the SRAMs can be totally or partially switched off to further reduced consumption. The BOR is always available in Stop 0 mode.

- Entry: The Stop mode is entered using the HAL_PWR_EnterSTOPMode() function with :
  - StopEntry:
    - PWR_STOPENTRY_WFI : Enter STOP mode with WFI instruction.
    - PWR_STOPENTRY_WFE : Enter STOP mode with WFE instruction and clear of pending events before.
    - PWR_STOPENTRY_WFE_NO_EVT_CLEAR: Enter STOP mode with WFE instruction and no clear of pending event before. -@@- The Regulator parameter is not used for the STM32U5 family and is kept as parameter just to maintain compatibility with other families.
- Exit: Any EXTI line configured in interrupt mode (the corresponding EXTI interrupt vector must be enabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Any peripheral interrupt occurring when the AHB/APB clocks are present due to an autonomous peripheral clock request (the peripheral vector must be enabled in the NVIC).

Standby mode

The Standby mode is used to achieve the lowest power consumption with BOR. The internal regulator is switched off so that the VCORE domain is powered off. The PLL, the MSI (MSIS and MSIK) RC, the HSI16 RC and the HSE crystal oscillators are also switched off. The RTC can remain active (Standby mode with RTC, Standby mode without RTC). The Brownout reset (BOR) always remains active in Standby mode. The state of each I/O during Standby mode can be selected by software: I/O with internal pull-up, internal pull-down or floating. After entering Standby mode, SRAMs and register contents are lost except for registers and backup SRAM in the Backup domain and Standby circuitry. Optionally, the full SRAM2 or 8 Kbytes or 56 Kbytes can be retained in Standby mode, supplied by the low-power regulator (Standby with RAM2 retention mode). The BORL (Brownout reset detector low) can be configured in ultra low power mode to further reduce power consumption during Standby mode. The device exits Standby mode upon an external reset (NRST pin), an IWDG reset, WKUP pin event (configurable rising or falling edge), an RTC event occurs (alarm, periodic wakeup, timestamp), or a tamper detection. The system clock after wakeup is MSIS up to 4 MHz. (++) Entry: The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function. (++) Exit: WKUPx pin edge, RTC event, external Reset in NRST pin, IWDG Reset, BOR reset.

This section contains the following APIs:
- *HAL_PWR_ConfigPVD()*
- *HAL_PWR_EnablePVD()*
- *HAL_PWR_DisablePVD()*
- *HAL_PWR_EnableWakeUpPin()*
- *HAL_PWR_DisableWakeUpPin()*
- *HAL_PWR_EnterSLEEPMode()*
- *HAL_PWR_EnterSTOPMode()*
- *HAL_PWR_EnterSTANDBYMode()*
- *HAL_PWR_EnableSleepOnExit()*
- *HAL_PWR_DisableSleepOnExit()*
- *HAL_PWR_EnableSEVOnPend()*
- *HAL_PWR_DisableSEVOnPend()*
- *HAL_PWR_PVD_IRQHandler()*
- *HAL_PWR_PVDCallback()*

61.2.5 PWR Attributes Functions

When the TrustZone security is activated by the TZEN option bit in the FLASH_OPTR register, some PWR register fields can be secured against non-secure access. The PWR TrustZone security allows the following features to be secured through the PWR_SECCFGR register : (++) Low-power mode. (++) Wake-up (WKUP) pins. (++) Voltage detection and monitoring. (++) VBAT mode. (++) I/Os pull-up/pull-down configuration. Other PWR configuration bits are secure when : (++) The system clock selection is secure in RCC: the voltage scaling (VOS) configuration and the regulator booster (BOOSTEN) are secure. (++) A GPIO is configured as secure: its corresponding bit for pull-up / pull-down configuration in Standby mode is secure. (++) The UCPD1 is secure in the GTZC: the PWR_UCPDR register is secure. A non-secure access to a secure-protected register bit is denied : (++) The secured bits are not written (WI) with a non-secure write access. (++) The secured bits are read as 0 (RAZ) with a non-secure read access.

When the TrustZone security is disabled (TZEN = 0), PWR_SECCFGR is RAZ/WI and all other registers are non-secure.

By default, after a reset, all PWR registers can be read or written with both privileged and unprivileged accesses, except PWR_PRIVCFGR that can be written with privileged access only. PWR_PRIVCFGR can be read by secure and non secure, privileged and unprivileged accesses. The SPRIV bit in PWR_PRIVCFGR can be written with secure privileged access only. This bit configures the privileged access of all PWR secure functions (defined by PWR_SECCFGR, GTZC, RCC or GPIO). When the SPRIV bit is set in PWR_PRIVCFGR: (++) The PWR secure bits can be written only with privileged access, including PWR_SECCFGR. (++) The PWR secure bits can be read only with privileged access except PWR_SECCFGR and PWR_PRIVCFGR that can be read by privileged or unprivileged access. (++) An unprivileged access to a privileged PWR bit or register is discarded : the bits are read as zero and the write to these bits is ignored (RAZ/WI). The NSPRIV bit of PWR_PRIVCFGR can be written with privileged access only, secure or non-secure. This bit configures the privileged access of all PWR securable functions that are configured as non-secure (defined by PWR_SECCFGR, GTZC, RCC or GPIO). When the NSPRIV bit is set in PWR_PRIVCFGR : (++) The PWR securable bits that are configured as non-secure, can be written only with privileged access. (++) The PWR securable bits that are configured as non-secure, can be read only with privileged access except PWR_PRIVCFGR that can be read by privileged or unprivileged accesses. (++) The VOSRDY and BOOSTRDY bits in PWR_VOSR, PWR_SR, PWR_SVMSR, PWR_BDSR and PWR_WUSR, can be read with privileged or unprivileged accesses. (++) An unprivileged access to a privileged PWR bit or register is discarded : the bits are read as zero and the write to these bits is ignored (RAZ/WI).

This section contains the following APIs:

- *HAL_PWR_ConfigAttributes()*
- *HAL_PWR_GetConfigAttributes()*

61.2.6 Detailed description of functions

HAL_PWR_DeInit

Function name

void HAL_PWR_DeInit (void)

Function description

Deinitialize the HAL PWR peripheral registers to their default reset values.

Return values

- **None.:**

Notes

- This functionality is not available in this product. The prototype is kept just to maintain compatibility with other products.

HAL_PWR_EnableBkUpAccess

Function name

void HAL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).

Return values

- **None.:**

Notes

- After a system reset, the backup domain is protected against possible unwanted write accesses.

HAL_PWR_DisableBkUpAccess

Function name

void HAL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain (RCC Backup domain control register RCC_BDCR, RTC registers, TAMP registers, backup registers and backup SRAM).

Return values

- **None.:**

HAL_PWR_ConfigPVD

Function name

HAL_StatusTypeDef HAL_PWR_ConfigPVD (PWR_PVDTypeDef * pConfigPVD)

Function description

Configure the voltage threshold detected by the Programmed Voltage Detector (PVD).

Parameters

- **pConfigPVD:** : Pointer to a PWR_PVDTypeDef structure that contains the PVD configuration information (PVDLevel and EventMode).

Return values

- **HAL:** Status.

HAL_PWR_EnablePVD

Function name

void HAL_PWR_EnablePVD (void)

Function description

Enable the programmable voltage detector (PVD).

Return values

- **None.:**

HAL_PWR_DisablePVD

Function name

void HAL_PWR_DisablePVD (void)

Function description

Disable the programmable voltage detector (PVD).

Return values

- **None.:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable the wake up line functionality.

Parameters

- **WakeUpPin:** : Specifies which wake up line to enable. This parameter can be one of PWR_WakeUp_Pins_High_Polarity define group where every param select the wake up line, the wake up source with high polarity detection and the wake up selected I/O or can be one of PWR_WakeUp_Pins_Low_Polarity define group where every param select the wake up line, the wake up source with low polarity and the wake up selected I/O or can be one of PWR_WakeUp_Pins define group where every param select the wake up line, the wake up source with high polarity and the first wake up I/O.

Return values

- **None.:**

Notes

- Wake up lines are used to wake up the system from Stop 3, Standby and Shutdown modes.

HAL_PWR_DisableWakeUpPin

Function name

void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the wake up line functionality.

Parameters

- **WakeUpPin:** : Specifies the wake up line to disable. This parameter can be a combination of all the following values :
  – PWR_WAKEUP_PIN1
  – PWR_WAKEUP_PIN2
  – PWR_WAKEUP_PIN3
  – PWR_WAKEUP_PIN4
  – PWR_WAKEUP_PIN6
  – PWR_WAKEUP_PIN5
  – PWR_WAKEUP_PIN7
  – PWR_WAKEUP_PIN8

Return values

- **None:**

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SleepEntry)

Function description

Enter the CPU in Sleep mode.

Parameters

- **Regulator:** : Specifies the regulator state in Sleep mode. This parameter can be one of the following values :
  – PWR_MAINREGULATOR_ON
  – PWR_LOWPOWERREGULATOR_ON
- **SleepEntry:** : Specifies if Sleep mode is entered with WFI or WFE instruction.
  – PWR_SLEEPENTRY_WFI : Enter SLEEP mode with WFI instruction.
  – PWR_SLEEPENTRY_WFE : Enter SLEEP mode with WFE instruction and clear of pending events before.
  – PWR_SLEEPENTRY_WFE_NO_EVT_CLEAR : Enter SLEEP mode with WFE instruction and no clear of pending event before.

Return values

- **None.:**

Notes

- In Sleep mode, all I/O pins keep the same state as in Run mode.
- CPU clock is off and all peripherals including Cortex-M33 core such as NVIC and SysTick can run and wake up the CPU when an interrupt or an event occurs.
- This parameter is not available in this product. The parameter is kept just to maintain compatibility with other products.

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t StopEntry)

Function description

Enter the whole system to Stop 0 mode.

Parameters

- **Regulator:** : Specifies the regulator state in Stop mode. This parameter can be one of the following values :
  – PWR_MAINREGULATOR_ON
  – PWR_LOWPOWERREGULATOR_ON
- **StopEntry:** : Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values :
  – PWR_STOPENTRY_WFI : Enter STOP mode with WFI instruction.
  – PWR_STOPENTRY_WFE : Enter STOP mode with WFE instruction and clear of pending events before.
  – PWR_STOPENTRY_WFE_NO_EVT_CLEAR : Enter STOP mode with WFE instruction and no clear of pending event before.

Return values

- **None.:**

Notes

- In Stop 0 mode, the regulator remains in main regulator mode, allowing a very fast wakeup time but with much higher consumption comparing to other Stop modes.
- Stop 0 offers the largest number of active peripherals and wakeup sources, a smaller wakeup time but a higher consumption. Stop mode achieves the lowest power consumption while retaining the content of SRAM and registers. All clocks in the VCORE domain are stopped. The PLL, the MSI (MSIS and MSIK) RC, the HSI16 RC and the HSE crystal oscillators are disabled. The LSE or LSI is still running.
- The system clock when exiting from Stop mode can be either MSIS up to 24 MHz or HSI16, depending on software configuration.
- This parameter is not available in this product. The parameter is kept just to maintain compatibility with other products.
- In System STOP mode, all I/O pins keep the same state as in Run mode.

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enter the whole system to Standby mode.

Return values

- **None.:**

Notes

- The Standby mode is used to achieve the lowest power consumption with BOR. The internal regulator is switched off so that the VCORE domain is powered off. The PLL, the MSI (MSIS and MSIK) RC, the HSI16 RC and the HSE crystal oscillators are also switched off.
- After entering Standby mode, SRAMs and register contents are lost except for registers and backup SRAM in the Backup domain and Standby circuitry. Optionally, the full SRAM2 or 8 Kbytes or 56 Kbytes can be retained in Standby mode, supplied by the low-power regulator (Standby with RAM2 retention mode) through HAL_PWREx_EnableSRAM2ContentStandbyRetention().
- The state of each I/O during Standby mode can be selected by software : I/O with internal pull-up through HAL_PWREx_EnableGPIOPullUp() and internal pull-down through HAL_PWREx_EnableGPIOPullDown().

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Indicate SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters Sleep mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name

void HAL_PWR_DisableSleepOnExit (void)

Function description

Disable SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters Sleep mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable CORTEX SEV-ON-PEND feature.

Return values

- **None.:**

Notes

- Sets SEVONPEND bit of SCR register. When this bit is set, any pending event / interrupt even if it's disabled or has insufficient priority to cause exception entry wakes up the Cortex-M33.

HAL_PWR_DisableSEVOnPend

Function name

void HAL_PWR_DisableSEVOnPend (void)

Function description

Disable CORTEX SEVONPEND feature.

Return values

- **None.:**

Notes

- Resets SEVONPEND bit of SCR register. When this bit is reset, only enabled pending event / interrupt to cause exception entry wakes up the Cortex-M33.

HAL_PWR_PVD_IRQHandler

Function name

void HAL_PWR_PVD_IRQHandler (void)

Function description

This function handles the PWR PVD interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the PVD_AVD_IRQHandler().

HAL_PWR_PVDCallback

Function name

void HAL_PWR_PVDCallback (void)

Function description

PWR PVD interrupt callback.

Return values

- **None.:**

HAL_PWR_ConfigAttributes

Function name

void HAL_PWR_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the PWR item attributes.

Parameters

- **Item:** : Specifies the item(s) to set attributes on. This parameter can be a combination of PWR Items.
- **Attributes:** : Specifies the available attribute(s). This parameter can be one of PWR Attributes.

Return values

- **None.:**

Notes

- Available attributes are security and privilege protection.
- Security attribute can only be set only by secure access.
- Privilege attribute for secure items can be managed only by a secure privileged access.
- Privilege attribute for nsecure items can be managed by a secure privileged access or by a nsecure privileged access.

HAL_PWR_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_PWR_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get attribute(s) of a PWR item.

Parameters

- **Item:** : Specifies the item(s) to get attributes of. This parameter can be one of PWR Items.
- **pAttributes:** : Pointer to return attribute(s). Returned value could be one of PWR Attributes.

Return values

- **HAL:** Status.

61.3 PWR Firmware driver defines

The following section lists the various define and macros of the module.

61.3.1 PWR

PWR

PWR Attributes

PWR_NSEC_PRIV

NSecure and Privileged attribute

PWR_NSEC_NPRIV

NSecure and NPrivileged attribute

PWR_SEC_PRIV

Secure and Privileged attribute

PWR_SEC_NPRIV

Secure and NPrivileged attribute

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check PWR flags are set or not.

Parameters:

- __FLAG__: : Specifies the flag to check. This parameter can be one of the following values :
  - PWR_FLAG_VOSRDY : Voltage scaling ready flag. Indicates that the Vcore level at or above VOS selected level.
  - PWR_FLAG_BOOSTRDY : EPOD booster ready flag. Indicates that EPOD booster ready, frequency could be higher than 50 MHz.
  - PWR_FLAG_USBBOOSTRDY : USB EPOD booster ready flag. Indicates that USB EPOD booster ready, frequency could be higher than 50 MHz. This flag is available only for STM32U59xxx and STM32U5Axxx devices.
  - PWR_FLAG_STOPF : Stop flag. Indicates that the device was resumed from Stop mode.
  - PWR_FLAG_SBF : Standby flag. Indicates that the device was resumed from Standby mode.
  - PWR_FLAG_VDDA2RDY : VDDA2 ready flag (versus 1.8 V threshold). Indicates that VDDA is equal or above the threshold of the VDDA voltage monitor 2 (around 1.8 V).
  - PWR_FLAG_VDDA1RDY : VDDA1 ready flag (versus 1.6 V threshold). Indicates that VDDA is equal or above the threshold of the VDDA voltage monitor 1 (around 1.6 V).
  - PWR_FLAG_VDDIO2RDY : VDDIO2 ready flag (versus 0.9 V threshold). Indicates that VDDIO2 is equal or above the threshold of the VDDIO2 voltage monitor (around 0.9 V).
  - PWR_FLAG_VDDUSBRDY : VDDUSB ready flag (versus 1.2 V threshold). Indicates that VDDUSB is equal or above the threshold of the VDDUSB voltage monitor (around 1.2 V).
  - PWR_FLAG_ACTVOSRDY : Currently applied VOS ready flag. Indicates that Vcore is equal to the current voltage scaling provided by ACTVOS.
  - PWR_FLAG_PVDO VDD : Voltage detector output flag. Indicates that Vdd is equal or above the PVD threshold selected by PVDLS.
  - PWR_FLAG_REGS : Regulator selection flag. Indicates the regulator selected.
  - PWR_FLAG_TEMPH : Temperature level flag (versus high threshold). Indicates the temperature is equal or above high threshold.
  - PWR_FLAG_TEMPL : Temperature level flag (versus low threshold). Indicates the temperature is equal or above low threshold.
  - PWR_FLAG_VBATH : Backup domain voltage level flag (versus high threshold). Indicates the backup domain voltage level is equal or above high threshold.
  - PWR_WAKEUP_FLAG1 : Wakeup flag 1. Indicates that a wakeup event was received from the WKUP line 1.
  - PWR_WAKEUP_FLAG2 : Wakeup flag 2. Indicates that a wakeup event was received from the WKUP line 2.
  - PWR_WAKEUP_FLAG3 : Wakeup flag 3. Indicates that a wakeup event was received from the WKUP line 3.
  - PWR_WAKEUP_FLAG4 : Wakeup flag 4. Indicates that a wakeup event was received from the WKUP line 4.
  - PWR_WAKEUP_FLAG5 : Wakeup flag 5. Indicates that a wakeup event was received from the WKUP line 5.
  - PWR_WAKEUP_FLAG6 : Wakeup flag 6. Indicates that a wakeup event was received from the WKUP line 6.
  - PWR_WAKEUP_FLAG7 : Wakeup flag 7. Indicates that a wakeup event was received from the WKUP line 7.
  - PWR_WAKEUP_FLAG8 : Wakeup flag 8. Indicates that a wakeup event was received from the WKUP line 8.

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_PWR_CLEAR_FLAG

Description:
- Clear PWR flags.

Parameters:
- __FLAG__: : Specifies the flag to clear. This parameter can be one of the following values :
  - PWR_FLAG_STOPF : Stop flag. Indicates that the device was resumed from Stop mode.
  - PWR_FLAG_SBF : Standby flag. Indicates that the device was resumed from Standby mode.
  - PWR_WAKEUP_FLAG1 : Wakeup flag 1. Indicates that a wakeup event was received from the WKUP line 1.
  - PWR_WAKEUP_FLAG2 : Wakeup flag 2. Indicates that a wakeup event was received from the WKUP line 2.
  - PWR_WAKEUP_FLAG3 : Wakeup flag 3. Indicates that a wakeup event was received from the WKUP line 3.
  - PWR_WAKEUP_FLAG4 : Wakeup flag 4. Indicates that a wakeup event was received from the WKUP line 4.
  - PWR_WAKEUP_FLAG5 : Wakeup flag 5. Indicates that a wakeup event was received from the WKUP line 5.
  - PWR_WAKEUP_FLAG6 : Wakeup flag 6. Indicates that a wakeup event was received from the WKUP line 6.
  - PWR_WAKEUP_FLAG7 : Wakeup flag 7. Indicates that a wakeup event was received from the WKUP line 7.
  - PWR_WAKEUP_FLAG8 : Wakeup flag 8. Indicates that a wakeup event was received from the WKUP line 8.

Return value:
- None.

__HAL_PWR_PVD_EXTI_ENABLE_IT

Description:
- Enable the PVD Extended Interrupt Line.

Return value:
- None.

__HAL_PWR_PVD_EXTI_DISABLE_IT

Description:
- Disable the PVD Extended Interrupt Line.

Return value:
- None.

__HAL_PWR_PVD_EXTI_ENABLE_EVENT

Description:
- Enable the PVD Event Line.

Return value:
- None.

__HAL_PWR_PVD_EXTI_DISABLE_EVENT

Description:
- Disable the PVD Event Line.

Return value:
- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE

Description:
- Enable the PVD Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE

Description:
- Disable the PVD Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE

Description:
- Enable the PVD Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE

Description:
- Disable the PVD Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE

Description:
- Enable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE

Description:
- Disable the PVD Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

Description:
- Generate a Software Interrupt on selected EXTI line.

Return value:
- None

__HAL_PWR_PVD_EXTI_GET_FLAG

Description:
- Check whether or not the PVD EXTI Interrupt flag is set.

Return value:
- EXTI: PVD Line Status.

__HAL_PWR_PVD_EXTI_CLEAR_FLAG

Description:
- Clear the PVD EXTI Interrupt flag.

Return value:
- None.

PWR Flags

PWR_FLAG_VOSRDY
Voltage scaling ready flag

PWR_FLAG_BOOSTRDY
EPOD booster ready flag

PWR_FLAG_STOPF
Stop flag

PWR_FLAG_SBF
Standby flag

PWR_FLAG_VDDA2RDY
VDDA ready flag (versus 1.8 V threshold)

PWR_FLAG_VDDA1RDY
VDDA ready flag (versus 1.6 V threshold)

PWR_FLAG_VDDIO2RDY
VDDIO2 ready flag

PWR_FLAG_VDDUSBRDY
VDDUSB ready flag

PWR_FLAG_ACTVOSRDY
Currently applied VOS ready flag

PWR_FLAG_PVDO
VDD voltage detector output flag

PWR_FLAG_REGS
Regulator selection flag

PWR_FLAG_TEMPH
Temperature level flag (versus high threshold)

PWR_FLAG_TEMPL
Temperature level flag (versus low threshold)

PWR_FLAG_VBATH
Backup domain voltage level flag (versus high threshold)

PWR_FLAG_USBBOOSTRDY
USB EPOD booster ready flag

PWR Items

PWR_WKUP1
WUP1 secure protection

PWR_WKUP2

WUP2 secure protection

PWR_WKUP3

WUP3 secure protection

PWR_WKUP4

WUP4 secure protection

PWR_WKUP5

WUP5 secure protection

PWR_WKUP6

WUP6 secure protection

PWR_WKUP7

WUP7 secure protection

PWR_WKUP8

WUP8 secure protection

PWR_LPM

Low power modes secure protection

PWR_VDM

Voltage detection and monitoring secure protection

PWR_VB

Backup domain secure protection

PWR_APC

Pull-up/pull-down secure protection

PWR_ALL

Programmable Voltage Detection Level

PWR_PVDLEVEL_0

PVD threshold around 2.0 V

PWR_PVDLEVEL_1

PVD threshold around 2.2 V

PWR_PVDLEVEL_2

PVD threshold around 2.4 V

PWR_PVDLEVEL_3

PVD threshold around 2.5 V

PWR_PVDLEVEL_4

PVD threshold around 2.6 V

PWR_PVDLEVEL_5

PVD threshold around 2.8 V

PWR_PVDLEVEL_6

PVD threshold around 2.9 V

PWR_PVDLEVEL_7

External input analog voltage (compared internally to VREFINT)

PWR PVD Mode

PWR_PVD_MODE_NORMAL

Basic Mode is used

PWR_PVD_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_PVD_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_PVD_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVD_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVD_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR PVD Mode Mask

PVD_RISING_EDGE

Mask for rising edge set as PVD trigger

PVD_FALLING_EDGE

Mask for falling edge set as PVD trigger

PVD_MODE_IT

Mask for interruption yielded by PVD threshold crossing

PVD_MODE_EVT

Mask for event yielded by PVD threshold crossing

PWR PVM Mode Mask

PVM_RISING_EDGE

Mask for rising edge set as PVM trigger

PVM_FALLING_EDGE

Mask for falling edge set as PVM trigger

PVM_MODE_IT

Mask for interruption yielded by PVM threshold crossing

PVM_MODE_EVT

Mask for event yielded by PVM threshold crossing

PWR Regulator State in Sleep/Stop Mode

PWR_MAINREGULATOR_ON

Main Regulator ON in Run Mode

PWR_LOWPOWERREGULATOR_ON

Main Regulator ON in Low Power Mode

PWR Sleep Mode Entry

PWR_SLEEPENTRY_WFI

Wait For Interruption instruction to enter Sleep mode

PWR_SLEEPENTRY_WFE

Wait For Event instruction to enter Sleep mode

PWR_SLEEPENTRY_WFE_NO_EVT_CLEAR

PWR Stop Mode Entry

PWR_STOPENTRY_WFI

Wait For Interruption instruction to enter Stop mode

PWR_STOPENTRY_WFE

Wait For Event instruction to enter Stop mode

PWR_STOPENTRY_WFE_NO_EVT_CLEAR

PWR Wakeup Lines Flags

PWR_WAKEUP_FLAG1

Wakeup flag 1

PWR_WAKEUP_FLAG2

Wakeup flag 2

PWR_WAKEUP_FLAG3

Wakeup flag 3

PWR_WAKEUP_FLAG4

Wakeup flag 4

PWR_WAKEUP_FLAG5

Wakeup flag 5

PWR_WAKEUP_FLAG6

Wakeup flag 6

PWR_WAKEUP_FLAG7

Wakeup flag 7

PWR_WAKEUP_FLAG8

Wakeup flag 8

PWR_WAKEUP_ALL_FLAG

Wakeup flag all

PWR Wake Up Pins (Default polarity and default pin selection)

PWR_WAKEUP_PIN1

PA0 : Wakeup pin 1 (high polarity)

PWR_WAKEUP_PIN2

PA4 : Wakeup pin 2 (high polarity)

PWR_WAKEUP_PIN3

PE6 : Wakeup pin 3 (high polarity)

PWR_WAKEUP_PIN4

PA2 : Wakeup pin 4 (high polarity)

PWR_WAKEUP_PIN5

PC5 : Wakeup pin 5 (high polarity)

PWR_WAKEUP_PIN6

PB5 : Wakeup pin 6 (high polarity)

PWR_WAKEUP_PIN7

PB15 : Wakeup pin 7 (high polarity)

PWR_WAKEUP_PIN8

PF2 : Wakeup pin 8 (high polarity)

PWR Wake Up Pins High Polarity

PWR_WAKEUP_PIN1_HIGH_0

PA0 : Wakeup pin 1 (high polarity)

PWR_WAKEUP_PIN1_HIGH_1

PB2 : Wakeup pin 1 (high polarity)

PWR_WAKEUP_PIN1_HIGH_2

PE4 : Wakeup pin 1 (high polarity)

PWR_WAKEUP_PIN2_HIGH_0

PA4 : Wakeup pin 2 (high polarity)

PWR_WAKEUP_PIN2_HIGH_1

PC13 : Wakeup pin 2 (high polarity)

PWR_WAKEUP_PIN2_HIGH_2

PE5 : Wakeup pin 2 (high polarity)

PWR_WAKEUP_PIN3_HIGH_0

PE6 : Wakeup pin 3 (high polarity)

PWR_WAKEUP_PIN3_HIGH_1

PA1 : Wakeup pin 3 (high polarity)

PWR_WAKEUP_PIN3_HIGH_2

PB6 : Wakeup pin 3 (high polarity)

PWR_WAKEUP_PIN4_HIGH_0

PA2 : Wakeup pin 4 (high polarity)

PWR_WAKEUP_PIN4_HIGH_1

PB1 : Wakeup pin 4 (high polarity)

PWR_WAKEUP_PIN4_HIGH_2

PB7 : Wakeup pin 4 (high polarity)

PWR_WAKEUP_PIN5_HIGH_0
PC5 : Wakeup pin 5 (high polarity)

PWR_WAKEUP_PIN5_HIGH_1
PA3 : Wakeup pin 5 (high polarity)

PWR_WAKEUP_PIN5_HIGH_2
PB8 : Wakeup pin 5 (high polarity)

PWR_WAKEUP_PIN6_HIGH_0
PB5 : Wakeup pin 6 (high polarity)

PWR_WAKEUP_PIN6_HIGH_1
PA5 : Wakeup pin 6 (high polarity)

PWR_WAKEUP_PIN6_HIGH_2
PE7 : Wakeup pin 6 (high polarity)

PWR_WAKEUP_PIN6_HIGH_3
RTC : Wakeup pin 6 (high polarity)

PWR_WAKEUP_PIN7_HIGH_0
PB15 : Wakeup pin 7 (high polarity)

PWR_WAKEUP_PIN7_HIGH_1
PA6 : Wakeup pin 7 (high polarity)

PWR_WAKEUP_PIN7_HIGH_2
PE8 : Wakeup pin 7 (high polarity)

PWR_WAKEUP_PIN7_HIGH_3
RTC : Wakeup pin 7 (high polarity)

PWR_WAKEUP_PIN8_HIGH_0
PF2 : Wakeup pin 8 (high polarity)

PWR_WAKEUP_PIN8_HIGH_1
PA7 : Wakeup pin 8 (high polarity)

PWR_WAKEUP_PIN8_HIGH_2
PB10 : Wakeup pin 8 (high polarity)

PWR_WAKEUP_PIN8_HIGH_3
RTC : Wakeup pin 8 (high polarity)

PWR Wake Up Pins Low Polarity

PWR_WAKEUP_PIN1_LOW_0
PA0 : Wakeup pin 1 (low polarity)

PWR_WAKEUP_PIN1_LOW_1
PB2 : Wakeup pin 1 (low polarity)

PWR_WAKEUP_PIN1_LOW_2
PE4 : Wakeup pin 1 (low polarity)

PWR_WAKEUP_PIN2_LOW_0
PA4 : Wakeup pin 2 (low polarity)

PWR_WAKEUP_PIN2_LOW_1

PC13 : Wakeup pin 2 (low polarity)

PWR_WAKEUP_PIN2_LOW_2

PE5 : Wakeup pin 2 (low polarity)

PWR_WAKEUP_PIN3_LOW_0

PE6 : Wakeup pin 3 (low polarity)

PWR_WAKEUP_PIN3_LOW_1

PA1 : Wakeup pin 3 (low polarity)

PWR_WAKEUP_PIN3_LOW_2

PB6 : Wakeup pin 3 (low polarity)

PWR_WAKEUP_PIN4_LOW_0

PA2 : Wakeup pin 4 (low polarity)

PWR_WAKEUP_PIN4_LOW_1

PB1 : Wakeup pin 4 (low polarity)

PWR_WAKEUP_PIN4_LOW_2

PB7 : Wakeup pin 4 (low polarity)

PWR_WAKEUP_PIN5_LOW_0

PC5 : Wakeup pin 5 (low polarity)

PWR_WAKEUP_PIN5_LOW_1

PA3 : Wakeup pin 5 (low polarity)

PWR_WAKEUP_PIN5_LOW_2

PB8 : Wakeup pin 5 (low polarity)

PWR_WAKEUP_PIN6_LOW_0

PB5 : Wakeup pin 6 (low polarity)

PWR_WAKEUP_PIN6_LOW_1

PA5 : Wakeup pin 6 (low polarity)

PWR_WAKEUP_PIN6_LOW_2

PE7 : Wakeup pin 6 (low polarity)

PWR_WAKEUP_PIN6_LOW_3

RTC : Wakeup pin 6 (low polarity)

PWR_WAKEUP_PIN7_LOW_0

PB15 : Wakeup pin 7 (low polarity)

PWR_WAKEUP_PIN7_LOW_1

PA6 : Wakeup pin 7 (low polarity)

PWR_WAKEUP_PIN7_LOW_2

PE8 : Wakeup pin 7 (low polarity)

PWR_WAKEUP_PIN7_LOW_3

RTC : Wakeup pin 7 (low polarity)

PWR_WAKEUP_PIN8_LOW_0

PF2 : Wakeup pin 8 (low polarity)

PWR_WAKEUP_PIN8_LOW_1

PA7 : Wakeup pin 8 (low polarity)

PWR_WAKEUP_PIN8_LOW_2

PB10 : Wakeup pin 8 (low polarity)

PWR_WAKEUP_PIN8_LOW_3

RTC : Wakeup pin 8 (low polarity)

62 HAL PWR Extension Driver

62.1 PWREx Firmware driver registers structures

62.1.1 PWR_PVMTypeDef

PWR_PVMTypeDef is defined in the stm32u5xx_hal_pwr_ex.h

Data Fields

- ***uint32_t PVMType***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVMTypeDef::PVMType***
  Specifies which voltage is monitored. This parameter can be a value of ***PWREx_PVM_Type***.
- ***uint32_t PWR_PVMTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***PWREx_PVM_Mode***.

62.2 PWREx Firmware driver API description

The following section lists the various functions of the PWREx library.

62.2.1 How to use this driver

1. Call HAL_PWREx_ControlVoltageScaling() and HAL_PWREx_GetVoltageRange() to set / get the voltage scaling range. (+) Voltage scaling can be one of the following values :
   – voltage output scale 1 : 1V2 => Used when system clock frequency is up to 160 MHz
   – voltage output scale 2 : 1V1 => Used when system clock frequency is up to 100 MHz
   – voltage output scale 3 : 1V0 => Used when system clock frequency is up to 50 MHz
   – voltage output scale 4 : 0V9 => Used when system clock frequency is up to 24 MHz
2. Call HAL_PWREx_EnableFastSoftStart() and HAL_PWREx_DisableFastSoftStart() to enable / disable the fast software startup for the current regulator.
3. Call HAL_PWREx_EnterSTOP1Mode() function to enter the whole system to Stop 1 mode. Wake-up from Stop 1 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility).
4. Call HAL_PWREx_EnterSTOP2Mode() function to enter the whole system to Stop 2 mode. Wake-up from Stop 2 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility).
5. Call HAL_PWREx_EnterSTOP3Mode() function to enter the whole system to Stop 3 mode. Wake-up from Stop 3 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility).
6. Call HAL_PWREx_EnterSHUTDOWNMode() function to enter the whole system in Shutdown mode. Wake-up from Shutdown mode can be following to an external reset (NRST pin), a WKUP pin event (configurable rising or falling edge), or an RTC event occurs (alarm, periodic wakeup, timestamp), or a tamper detection.
7. Call HAL_PWREx_ConfigSRDDomain() to force in Run mode or to enter in Stop mode Smart Run Domain when the system enters Stop mode (Stop 0/1/2).
8. Call HAL_PWREx_EnableUltraLowPowerMode() and HAL_PWREx_DisableUltraLowPowerMode() to enable / disable the BOR ultra low power mode.
9. Call HAL_PWREx_S3WU_IRQHandler() function to handle the PWR Stop 3 wake up interrupt request.
10. Call HAL_PWREx_EnableBatteryCharging() and HAL_PWREx_DisableBatteryCharging() to enable / disable the battery charging capability when VDD alimentation is available.

11. Call HAL_PWREx_EnableVddUSB(), HAL_PWREx_EnableVddIO2() and HAL_PWREx_EnableVddA() to enable respectively VDDUSB, VDDIO2 and VDDA electrical and logical isolation. It is recommended to disable VDDUSB, VDDIO2 and VDDA electrical and logical isolation through HAL_PWREx_DisableVddUSB(), HAL_PWREx_DisableVddIO2() and HAL_PWREx_DisableVddA().
12. Call HAL_PWREx_ConfigPVM() after setting parameters to be configured (event mode and PVD type) in order to set up the Peripheral Voltage Monitor, then use HAL_PWREx_EnableUVM(), HAL_PWREx_EnableIO2VM(), HAL_PWREx_EnableAVM1() and HAL_PWREx_EnableAVM2() functions to start the PVM VDDx monitoring and use HAL_PWREx_DisableUVM(), HAL_PWREx_DisableIO2VM(), HAL_PWREx_DisableAVM1() and HAL_PWREx_DisableAVM2() to stop the PVM VDDx monitoring. (+) PVM monitored voltages are :
    – VDDUSB versus 1V2
    – VDDIO2 versus 0V9
    – VDDA versus 1V6
    – VDDA versus 1V8
13. Call HAL_PWREx_EnableUSBHSTranceiverSupply() and HAL_PWREx_DisableUSBHSTranceiverSupply() to enable / disable the internal USB HS transceiver supply. (+) This feature is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices
14. Call HAL_PWREx_EnableOTGHSPHYLowPowerRetention() and HAL_PWREx_DisableOTGHSPHYLowPowerRetention() to enable / disable OTG_HS PHY power during low power modes (Stop2, Stop3 and Standby). (+) This feature is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices
15. Call HAL_PWREx_EnableVDD11USB() and HAL_PWREx_DisableVDD11USB() to enable/ disable the VDD11USB. (+) This feature is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices
16. Call HAL_PWREx_EnableMonitoring() and HAL_PWREx_DisableMonitoring() to enable / disable the VBAT and temperature monitoring.
17. Call HAL_PWREx_EnableUCPDStandbyMode() and HAL_PWREx_DisableUCPDStandbyMode() to enable / disable the UCPD configuration memorization in Standby mode.
18. Call HAL_PWREx_EnableUCPDDeadBattery() and HAL_PWREx_DisableUCPDDeadBattery() to enable / disable the dead battery behavior.
19. Call HAL_PWREx_PVD_PVM_IRQHandler() function to handle the PWR PVD and PVM interrupt request.
20. Call HAL_PWREx_EnableSRAM2ContentStandbyRetention() and HAL_PWREx_DisableSRAM2ContentStandbyRetention() to enable / disable the SRAM2 content retention in Stop 3 and Standby low power modes.

21. Call HAL_PWREx_EnableRAMsContentStopRetention() and HAL_PWREx_DisableRAMsContentStopRetention() to enable / disable the RAMs content retention in Stop mode (Stop 0/1/2/3). (+) Retained RAM can be one of the following RAMs :
    - SRAM1
    - SRAM2
    - SRAM3 (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - SRAM4
    - SRAM5 (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - SRAM6 (available only for STM32U5Fxxx and STM32U5Gxxx devices)
    - ICACHE
    - DMA2DRAM (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - PKA32RAM
    - DCACHE1
    - FMAC
    - FDCAN
    - USB
    - DCACHE2 (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - LTDC (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - GFXMMU (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - DSI (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - JPEG (available only for STM32U5Fxxx and STM32U5Gxxx devices)
22. Call HAL_PWREx_EnableRAMsContentRunRetention() and HAL_PWREx_DisableRAMsContentRunRetention() to enable / disable the RAMs content retention in Run mode. (+) Retained RAM can be one of the following RAMs :
    - SRAM1
    - SRAM2
    - SRAM3 (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - SRAM4
    - SRAM5 (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
    - SRAM6 (available only for STM32U5Fxxx and STM32U5Gxxx devices)
23. Call HAL_PWREx_EnableFlashFastWakeUp() and HAL_PWREx_DisableFlashFastWakeUp() to enable / disable the flash memory fast wakeup from Stop mode (Stop 0/1).
24. Call HAL_PWREx_EnableSRAM4FastWakeUp() and HAL_PWREx_DisableSRAM4FastWakeUp() to enable / disable the SRAM4 memory fast wakeup from Stop mode (Stop 0/1/2).
25. Call HAL_PWREx_EnableBkupRAMRetention() and HAL_PWREx_DisableBkupRAMRetention() to enable / disable the Backup RAM content retention in Standby, Shutdown and VBAT modes.
26. Call HAL_PWREx_EnablePullUpPullDownConfig() and HAL_PWREx_DisablePullUpPullDownConfig() to I/O enable / disable pull-up and pull-down configuration.
27. Call HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() to apply respectively pull-up and pull-down to selected I/O. Call HAL_PWREx_DisableGPIOPullUp() and HAL_PWREx_DisableGPIOPullDown() to disable applied respectively pull-up and pull-down to selected I/O.

(#) Call HAL_PWREx_ControlVoltageScaling() and HAL_PWREx_GetVoltageRange() to set / get the voltage scaling range.

- Voltage scaling can be one of the following values :
  - voltage output scale 1 : 1V2 => Used when system clock frequency is up to 160 MHz
  - voltage output scale 2 : 1V1 => Used when system clock frequency is up to 100 MHz
  - voltage output scale 3 : 1V0 => Used when system clock frequency is up to 50 MHz
  - voltage output scale 4 : 0V9 => Used when system clock frequency is up to 24 MHz (#) Call HAL_PWREx_EnableFastSoftStart() and HAL_PWREx_DisableFastSoftStart() to enable / disable the fast software startup for the current regulator. (#) Call HAL_PWREx_EnterSTOP1Mode() function to enter the whole system to Stop 1 mode. Wake-up from Stop 1 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility). (#) Call HAL_PWREx_EnterSTOP2Mode() function to enter the whole system to Stop 2 mode. Wake-up from Stop 2 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility). (#) Call HAL_PWREx_EnterSTOP3Mode() function to enter the whole system to Stop 3 mode. Wake-up from Stop 3 mode could be following to an event or an interrupt according to low power mode intrinsic request called (__WFI() or __WFE()). (Regulator state on U5 devices is managed internally but regulator parameter is kept for product compatibility). (#) Call HAL_PWREx_EnterSHUTDOWNMode() function to enter the whole system in Shutdown mode. Wake-up from Shutdown mode can be following to an external reset (NRST pin), a WKUP pin event (configurable rising or falling edge), or an RTC event occurs (alarm, periodic wakeup, timestamp), or a tamper detection. (#) Call HAL_PWREx_ConfigSRDDomain() to force in Run mode or to enter in Stop mode Smart Run Domain when the system enters Stop mode (Stop 0/1/2). (#) Call HAL_PWREx_EnableUltraLowPowerMode() and HAL_PWREx_DisableUltraLowPowerMode() to enable / disable the BOR ultra low power mode. (#) Call HAL_PWREx_S3WU_IRQHandler() function to handle the PWR Stop 3 wake up interrupt request. (#) Call HAL_PWREx_EnableBatteryCharging() and HAL_PWREx_DisableBatteryCharging() to enable / disable the battery charging capability when VDD alimentation is available. (#) Call HAL_PWREx_EnableVddUSB(), HAL_PWREx_EnableVddIO2() and HAL_PWREx_EnableVddA() to enable respectively VDDUSB, VDDIO2 and VDDA electrical and logical isolation. It is recommended to disable VDDUSB, VDDIO2 and VDDA electrical and logical isolation through HAL_PWREx_DisableVddUSB(), HAL_PWREx_DisableVddIO2() and HAL_PWREx_DisableVddA(). (#) Call HAL_PWREx_ConfigPVM() after setting parameters to be configured (event mode and PVD type) in order to set up the Peripheral Voltage Monitor, then use HAL_PWREx_EnableUVM(), HAL_PWREx_EnableIO2VM(), HAL_PWREx_EnableAVM1() and HAL_PWREx_EnableAVM2() functions to start the PVM VDDx monitoring and use HAL_PWREx_DisableUVM(), HAL_PWREx_DisableIO2VM(), HAL_PWREx_DisableAVM1() and HAL_PWREx_DisableAVM2() to stop the PVM VDDx monitoring.
- PVM monitored voltages are :
  - VDDUSB versus 1V2
  - VDDIO2 versus 0V9
  - VDDA versus 1V6
  - VDDA versus 1V8 (#) Call HAL_PWREx_EnableUSBHSTranceiverSupply() and HAL_PWREx_DisableUSBHSTranceiverSupply() to enable / disable the internal USB HS transceiver supply.
- This feature is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices (#) Call HAL_PWREx_EnableOTGHSPHYLowPowerRetention() and HAL_PWREx_DisableOTGHSPHYLowPowerRetention() to enable / disable OTG_HS PHY power during low power modes (Stop2, Stop3 and Standby).
- This feature is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices (#) Call HAL_PWREx_EnableVDD11USB() and HAL_PWREx_DisableVDD11USB() to enable/ disable the VDD11USB.

- This feature is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices (#) Call HAL_PWREx_EnableMonitoring() and HAL_PWREx_DisableMonitoring() to enable / disable the VBAT and temperature monitoring. (#) Call HAL_PWREx_EnableUCPDStandbyMode() and HAL_PWREx_DisableUCPDStandbyMode() to enable / disable the UCPD configuration memorization in Standby mode. (#) Call HAL_PWREx_EnableUCPDDeadBattery() and HAL_PWREx_DisableUCPDDeadBattery() to enable / disable the dead battery behavior. (#) Call HAL_PWREx_PVD_PVM_IRQHandler() function to handle the PWR PVD and PVM interrupt request. (#) Call HAL_PWREx_EnableSRAM2ContentStandbyRetention() and HAL_PWREx_DisableSRAM2ContentStandbyRetention() to enable / disable the SRAM2 content retention in Stop 3 and Standby low power modes. (#) Call HAL_PWREx_EnableRAMsContentStopRetention() and HAL_PWREx_DisableRAMsContentStopRetention() to enable / disable the RAMs content retention in Stop mode (Stop 0/1/2/3).
- Retained RAM can be one of the following RAMs :
  - SRAM1
  - SRAM2
  - SRAM3 (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - SRAM4
  - SRAM5 (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - SRAM6 (available only for STM32U5Fxxx and STM32U5Gxxx devices)
  - ICACHE
  - DMA2DRAM (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - PKA32RAM
  - DCACHE1
  - FMAC
  - FDCAN
  - USB
  - DCACHE2 (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - LTDC (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - GFXMMU (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - DSI (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - JPEG (available only for STM32U5Fxxx and STM32U5Gxxx devices) (#) Call HAL_PWREx_EnableRAMsContentRunRetention() and HAL_PWREx_DisableRAMsContentRunRetention() to enable / disable the RAMs content retention in Run mode.

- Retained RAM can be one of the following RAMs :
  - SRAM1
  - SRAM2
  - SRAM3 (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - SRAM4
  - SRAM5 (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - SRAM6 (available only for STM32U5Fxxx and STM32U5Gxxx devices) (#) Call HAL_PWREx_EnableFlashFastWakeUp() and HAL_PWREx_DisableFlashFastWakeUp() to enable / disable the flash memory fast wakeup from Stop mode (Stop 0/1). (#) Call HAL_PWREx_EnableSRAM4FastWakeUp() and HAL_PWREx_DisableSRAM4FastWakeUp() to enable / disable the SRAM4 memory fast wakeup from Stop mode (Stop 0/1/2). (#) Call HAL_PWREx_EnableBkupRAMRetention() and HAL_PWREx_DisableBkupRAMRetention() to enable / disable the Backup RAM content retention in Standby, Shutdown and VBAT modes. (#) Call HAL_PWREx_EnablePullUpPullDownConfig() and HAL_PWREx_DisablePullUpPullDownConfig() to I/O enable / disable pull-up and pull-down configuration. (#) Call HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() to apply respectively pull-up and pull-down to selected I/O. Call HAL_PWREx_DisableGPIOPullUp() and HAL_PWREx_DisableGPIOPullDown() to disable applied respectively pull-up and pull-down to selected I/O.

62.2.2 Power supply control functions

This section provides functions allowing to control power supply.

- The STM32U5 Series devices embed two regulators : one LDO (linear voltage regulator) and one SMPS (step down converter) in parallel to provide the VCORE supply for digital peripherals, SRAM1, SRAM2, SRAM3 and SRAM4 and embedded Flash memory.
- The SMPS allows the power consumption to be reduced but some peripherals can be perturbed by the noise generated by the SMPS, requiring the application to switch to LDO when running this peripheral in order to reach the best performances.
- The LDO and the SMPS regulators have two modes: Main regulator mode (used when performance is needed), and Low-power regulator mode. LDO or SMPS can be used in all voltage scaling ranges, and in all Stop modes.
- After reset, the regulator is the LDO, in Range 4. Switching to SMPS provides lower consumption in particular at high VDD voltage. It is possible to switch from LDO to SMPS, or from SMPS to LDO on the fly in any range, by configuring the REGSEL bit. It is recommended to switch first to SMPS before changing the voltage range.
- When exiting the Stop or Standby modes, the regulator is the same than when entering low power modes. The voltage range is the Range 4.
- Both regulators can provide four different voltages (voltage scaling) and can operate in Stop modes. Voltage scaling ranges can be one of the following values :
  - voltage output scale 1 : 1V2 => Used when system clock frequency is up to 160 MHz
  - voltage output scale 2 : 1V1 => Used when system clock frequency is up to 100 MHz
  - voltage output scale 3 : 1V0 => Used when system clock frequency is up to 50 MHz
  - voltage output scale 4 : 0V9 => Used when system clock frequency is up to 24 MHz

This section contains the following APIs:

- *HAL_PWREx_ControlVoltageScaling()*
- *HAL_PWREx_GetVoltageRange()*
- *HAL_PWREx_ConfigSupply()*
- *HAL_PWREx_GetSupplyConfig()*
- *HAL_PWREx_EnableFastSoftStart()*
- *HAL_PWREx_DisableFastSoftStart()*

62.2.3 Low power control functions

This section provides functions allowing to control low power modes.

Low Power modes configuration

This section presents 4 principles low power modes :
- Stop 1 mode : Cortex-M33 is stopped, clocks are stopped and the regulator is in low power mode. Several peripheral can operate in this mode.
- Stop 2 mode : Cortex-M33 is stopped, clocks are stopped and the regulator is in low power mode. Only autonomous peripherals can operate in this mode.
- Stop 3 mode : Cortex-M33 is stopped, clocks are stopped and the regulator is in low power mode. No peripheral can operate in this mode. Only RAMs content is preserved.
- Shutdown mode : All PWR domains enter Shutdown mode and the VCORE supply regulator is powered off. The SRAMs and register contents are lost except for registers in the Backup domain.

Stop 1 mode

The Stop 1 mode is based on the Cortex-M33 Deepsleep mode combined with the peripheral clock gating. The voltage regulator is configured in low power mode. In Stop 1 mode, all clocks in the VCORE domain are stopped. The PLL, MSIS, MSIK, HSI16 and HSE oscillators are disabled. Some peripherals with the LPBAM capability can switch on HSI16 or MSIS or MSIK for transferring data. All SRAMs and register contents are preserved, but the SRAMs can be totally or partially switched off to further reduced consumption. The BOR is always available in Stop 1 mode.
- Entry: The Stop 1 mode is entered by using the HAL_PWREx_EnterSTOP1Mode() function.
  – PWR_STOPENTRY_WFI: enter Stop 1 mode with WFI instruction.
  – PWR_STOPENTRY_WFE: enter Stop 1 mode with WFE instruction.
- Exit: Any EXTI line configured in interrupt mode (the corresponding EXTI interrupt vector must be enabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Any peripheral interrupt occurring when the AHB/APB clocks are present due to an autonomous peripheral clock request (the peripheral vector must be enabled in the NVIC) Any EXTI line configured in event mode.

Stop 2 mode

The Stop 2 mode is based on the Cortex-M33 Deepsleep mode combined with peripheral clock gating. In Stop 2 mode, all clocks in the VCORE domain are stopped. The PLL, MSIS, MSIK, HSI16 and HSE oscillators are disabled. Some peripherals with the LPBAM capability can switch on HSI16 or MSIS or MSIK for transferring data. All SRAMs and register contents are preserved, but the SRAMs can be totally or partially switched off to further reduce consumption. The BOR is always available in Stop 2 mode.
- Entry: The Stop 2 mode is entered by using the HAL_PWREx_EnterSTOP2Mode() function.
  – PWR_STOPENTRY_WFI: enter Stop 2 mode with WFI instruction.
  – PWR_STOPENTRY_WFE: enter Stop 2 mode with WFE instruction.
- Exit: Any EXTI line configured in interrupt mode (the corresponding EXTI interrupt vector must be enabled in the NVIC). The interrupt source can be external interrupts or peripherals with wakeup capability. Any peripheral interrupt occurring when the AHB/APB clocks are present due to an autonomous peripheral clock request (the peripheral vector must be enabled in the NVIC) Any EXTI line configured in event mode.

Stop 3 mode

The Stop 3 mode is based on the Cortex-M33 Deepsleep mode combined with peripheral clock gating. In Stop 3 mode, all clocks in the VCORE domain are stopped. The PLL, MSIS, MSIK, HSI16 and HSE oscillators are disabled. All SRAMs and register contents are preserved, but the SRAMs can be totally or partially switched off to further reduce consumption. The BOR is always available in Stop 3 mode.
- Entry: The Stop 3 mode is entered by using the HAL_PWREx_EnterSTOP3Mode() function.
  – PWR_STOPENTRY_WFI: enter Stop 3 mode with WFI instruction.
  – PWR_STOPENTRY_WFE: enter Stop 3 mode with WFE instruction.
- Exit: WKUPx pin edge, RTC or TAMP event, external Reset in NRST pin, IWDG Reset, BOR reset.

Shutdown mode

The lowest power consumption is reached in Shutdown mode. It is based on the Deepsleep mode with the voltage regulator disabled. The VCORE domain is consequently powered off. The PLL, HSI16, MSIS, MSIK and HSE oscillators are also switched off. The SRAMs and register contents are lost except for registers in the Backup domain. The BOR is not available in Shutdown mode. No power voltage monitoring is possible in this mode, therefore the switch to Backup domain is not supported.

- Entry: The Shutdown mode is entered by using the HAL_PWREx_EnterSHUTDOWNMode() function.
- Exit: WKUPx pin edge, RTC/TAMP event, external Reset in NRST pin.

This section contains the following APIs:

- *HAL_PWREx_EnterSTOP1Mode()*
- *HAL_PWREx_EnterSTOP2Mode()*
- *HAL_PWREx_EnterSTOP3Mode()*
- *HAL_PWREx_EnterSHUTDOWNMode()*
- *HAL_PWREx_ConfigSRDDomain()*
- *HAL_PWREx_EnableUltraLowPowerMode()*
- *HAL_PWREx_DisableUltraLowPowerMode()*
- *HAL_PWREx_S3WU_IRQHandler()*
- *HAL_PWREx_S3WUCallback()*

62.2.4 Voltage Monitoring Functions

This section provides functions allowing voltage monitoring.

PVM configuration

- The supplies (VDDA, VDDIO2 and VDDUSB) can be independent from VDD and can be monitored with four peripheral voltage monitoring (PVM):
  - The UVM monitors the USB supply VDDUSB. VDDUSBRDY indicates if the VDDUSB independent power supply is higher or lower than the VUVM threshold.
  - The IO2VM monitors the PG[15:2] supply VDDIO2. VDDIO2RDY indicates if the VDDIO2 independent power supply is higher or lower than the VIO2VM threshold.
  - The AVM1 monitors the analog supply VDDA. VDDA1RDY indicates if the VDDA independent power supply is higher or lower than the VAVM1 threshold.
  - The AVM2 monitors the analog supply VDDA. VDDA2RDY indicates if the VDDA independent power supply is higher or lower than the VAVM2 threshold.
- Each PVM output is connected to an EXTI line and can generate an interrupt if enabled through the EXTI registers. The PVMx output interrupt is generated when the independent power supply drops below the PVM threshold and/or when it rises above the PVM threshold, depending on EXTI line rising/falling edge configuration.
- Each PVM can remain active in Stop 0, Stop 1, Stop 2 modes, and the PVM interrupt can wake up from the Stop mode. The PVM is not functional in Stop 3 mode.

VBAT charging

When VDD is present, it is possible to charge the external battery on VBAT through an internal resistance. The VBAT charging is done either through a 5 kOhm resistor or through a 1.5 kOhm resistor depending on the VBRS bit value in the PWR_BDCR2 register. The battery charging is enabled by setting VBE bit in the PWR_BDCR2 register. It is automatically disabled in VBAT mode.

Backup domain monitoring

When the Backup domain voltage and temperature monitoring is enabled (MONEN = 1 in the PWR_BDCR1 register), the Backup domain voltage and the temperature are monitored. If the Backup domain voltage monitoring internal tamper is enabled in the TAMP peripheral (ITAMP1E = 1 in the TAMP_CR1 register), a tamper event is generated when the Backup domain voltage is above the functional range. In case the Backup domain voltage is below the functional range, a Brownout reset is generated, erasing all device including Backup domain.

Backup domain battery

- To retain the content of the backup registers and supply the RTC function when VDD is turned off, the VBAT pin can be connected to an optional backup voltage supplied by a battery or by another source. The VBAT pin powers the RTC unit, the LSE oscillator and the PC13 to PC15 I/Os, allowing the RTC to operate even when the main power supply is turned off. The backup SRAM is optionally powered by VBAT pin when the BREN bit is set in the PWR Backup domain control register 1 (PWR_BDCR1). The switch to the VBAT supply is controlled by the power down reset embedded in the Reset block.
- After exiting reset, the USB Type-C (dead battery) behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to handover control to the UCPD (the UCPD must be initialized before doing the disable).

This section contains the following APIs:
- *HAL_PWREx_ConfigPVM()*
- *HAL_PWREx_EnableVddUSB()*
- *HAL_PWREx_DisableVddUSB()*
- *HAL_PWREx_EnableVddIO2()*
- *HAL_PWREx_DisableVddIO2()*
- *HAL_PWREx_EnableVddA()*
- *HAL_PWREx_DisableVddA()*
- *HAL_PWREx_EnableUVM()*
- *HAL_PWREx_DisableUVM()*
- *HAL_PWREx_EnableIO2VM()*
- *HAL_PWREx_DisableIO2VM()*
- *HAL_PWREx_EnableAVM1()*
- *HAL_PWREx_DisableAVM1()*
- *HAL_PWREx_EnableAVM2()*
- *HAL_PWREx_DisableAVM2()*
- *HAL_PWREx_EnableMonitoring()*
- *HAL_PWREx_DisableMonitoring()*
- *HAL_PWREx_EnableUSBHSTranceiverSupply()*
- *HAL_PWREx_DisableUSBHSTranceiverSupply()*
- *HAL_PWREx_EnableOTGHSPHYLowPowerRetention()*
- *HAL_PWREx_DisableOTGHSPHYLowPowerRetention()*
- *HAL_PWREx_EnableVDD11USB()*
- *HAL_PWREx_DisableVDD11USB()*
- *HAL_PWREx_EnableUCPDStandbyMode()*
- *HAL_PWREx_DisableUCPDStandbyMode()*
- *HAL_PWREx_EnableUCPDDeadBattery()*
- *HAL_PWREx_DisableUCPDDeadBattery()*
- *HAL_PWREx_EnableBatteryCharging()*
- *HAL_PWREx_DisableBatteryCharging()*
- *HAL_PWREx_PVD_PVM_IRQHandler()*
- *HAL_PWREx_UVMCallback()*
- *HAL_PWREx_IO2VMCallback()*
- *HAL_PWREx_AVM1Callback()*
- *HAL_PWREx_AVM2Callback()*

62.2.5 Memories Retention Functions

Several STM32U5 devices RAMs are configurable to keep / lose RAMs content during Stop mode (Stop 0/1/2/3).

- Retained content RAMs in Stop modes are :
  - SRAM1
  - SRAM2
  - SRAM3 (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - SRAM4
  - ICACHE
  - DMA2DRAM (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - PKA32RAM
  - DCACHE
  - FMAC
  - FDCAN
  - USB

Several STM32U5 devices RAMs are configurable to keep / lose RAMs content during Run mode.

- Retained content RAMs in Run modes are :
  - SRAM1
  - SRAM2
  - SRAM3 (available only for STM32U575xx, STM32U585xx, STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices)
  - SRAM4

SRAM2 is configurable to keep / lose its content in Stop 3 and Standby modes. The SRAM2 retention feature granularity is at pages level.

BKPRAM is configurable to keep / lose its content in Standby, Shutdown and VBAT modes.

This section contains the following APIs:

- *HAL_PWREx_EnableSRAM2ContentStandbyRetention()*
- *HAL_PWREx_DisableSRAM2ContentStandbyRetention()*
- *HAL_PWREx_EnableRAMsContentStopRetention()*
- *HAL_PWREx_DisableRAMsContentStopRetention()*
- *HAL_PWREx_EnableRAMsContentRunRetention()*
- *HAL_PWREx_DisableRAMsContentRunRetention()*
- *HAL_PWREx_EnableBkupRAMRetention()*
- *HAL_PWREx_DisableBkupRAMRetention()*
- *HAL_PWREx_EnableFlashFastWakeUp()*
- *HAL_PWREx_DisableFlashFastWakeUp()*
- *HAL_PWREx_EnableSRAM4FastWakeUp()*
- *HAL_PWREx_DisableSRAM4FastWakeUp()*

62.2.6 IO Pull-Up Pull-Down Configuration Functions

In Standby and Shutdown mode, pull up and pull down can be configured to maintain an I/O in the selected state. If the APC bit in the PWR_APCR register is set, the I/Os can be configured either with a pull-up through PWR_PUCRx registers (x=A,B,C,D,E,F,G,H,I,J), or with a pull-down through PWR_PDCRx registers (x=A,B,C,D,E,F,G,H,I,J)), or can be kept in analog state if none of the PWR_PUCRx or PWR_PDCRx register is set.

- Port J is available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices.

The pull-down configuration has highest priority over pull-up configuration in case both PWR_PUCRx and PWR_PDCRx are set for the same I/O. This configuration is lost when exiting the Shutdown but not from Standby mode.

This section contains the following APIs:

- *HAL_PWREx_EnablePullUpPullDownConfig()*
- *HAL_PWREx_DisablePullUpPullDownConfig()*

- *HAL_PWREx_EnableGPIOPullUp()*
- *HAL_PWREx_DisableGPIOPullUp()*
- *HAL_PWREx_EnableGPIOPullDown()*
- *HAL_PWREx_DisableGPIOPullDown()*

62.2.7 Detailed description of functions

HAL_PWREx_ControlVoltageScaling

Function name

HAL_StatusTypeDef HAL_PWREx_ControlVoltageScaling (uint32_t VoltageScaling)

Function description

Configure the main internal regulator output voltage to achieve a tradeoff between performance and power consumption.

Parameters

- **VoltageScaling:** : Specifies the regulator output voltage scale. This parameter can be one of the following values :
  - PWR_REGULATOR_VOLTAGE_SCALE1 : Regulator voltage output scale 1. Provides a typical output voltage at 1.2 V. Used when system clock frequency is up to 160 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2 : Regulator voltage output scale 2. Provides a typical output voltage at 1.1 V. Used when system clock frequency is up to 100 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE3 : Regulator voltage output scale 3. Provides a typical output voltage at 1.0 V. Used when system clock frequency is up to 50 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE4 : Regulator voltage output scale 4. Provides a typical output voltage at 0.9 V. Used when system clock frequency is up to 24 MHz.

Return values

- **HAL:** Status.

Notes

- Before moving to voltage scaling 2, it is mandatory to ensure that the system frequency is between 50 MHz and 100 MHz.
- Before moving to voltage scaling 3, it is mandatory to ensure that the system frequency is between 24 MHz and 50 MHz.
- Before moving to voltage scaling 4, it is mandatory to ensure that the system frequency is below 24 MHz.

HAL_PWREx_GetVoltageRange

Function name

uint32_t HAL_PWREx_GetVoltageRange (void)

Function description

Return Voltage Scaling Range.

Return values

- **Applied:** voltage scaling value.

HAL_PWREx_ConfigSupply

Function name

HAL_StatusTypeDef HAL_PWREx_ConfigSupply (uint32_t SupplySource)

Function description

Configure the system Power Supply.

Parameters

- **SupplySource:** : Specifies the Power Supply source to set after a system startup. This parameter can be one of the following values :
  – PWR_LDO_SUPPLY : The LDO regulator supplies the Vcore Power Domains.
  – PWR_SMPS_SUPPLY : The SMPS regulator supplies the Vcore Power Domains.

Return values

- **HAL:** Status.

HAL_PWREx_GetSupplyConfig

Function name

uint32_t HAL_PWREx_GetSupplyConfig (void)

Function description

Get the power supply configuration.

Return values

- **The:** supply configured.

HAL_PWREx_EnableFastSoftStart

Function name

void HAL_PWREx_EnableFastSoftStart (void)

Function description

Enable fast soft start for the current regulator.

Return values

- **None.:**

HAL_PWREx_DisableFastSoftStart

Function name

void HAL_PWREx_DisableFastSoftStart (void)

Function description

Disable fast soft start for the current regulator.

Return values

- **None.:**

HAL_PWREx_EnterSTOP1Mode

Function name

void HAL_PWREx_EnterSTOP1Mode (uint8_t STOPEntry)

Function description

Enter the whole system to Stop 1 mode.

Parameters

- **STOPEntry:** : Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values :
  – PWR_STOPENTRY_WFI enter Stop mode with Wait For Interrupt request.
  – PWR_STOPENTRY_WFE enter Stop mode with Wait For Event request.

Return values

- **None.:**

Notes

- Stop 1 offers the largest number of active peripherals and wakeup sources, a smaller wakeup time but a higher consumption.
- Stop mode achieves the lowest power consumption while retaining the content of SRAM and registers. All clocks in the VCORE domain are stopped. The PLL, the MSI (MSIS and MSIK) RC, the HSI16 RC and the HSE crystal oscillators are disabled. The LSE or LSI is still running.
- The system clock when exiting from Stop mode can be either MSIS up to 24 MHz or HSI16, depending on software configuration.

HAL_PWREx_EnterSTOP2Mode

Function name

void HAL_PWREx_EnterSTOP2Mode (uint8_t STOPEntry)

Function description

Enter the whole system to Stop 2 mode.

Parameters

- **STOPEntry:** : Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values :
  - PWR_STOPENTRY_WFI enter Stop mode with Wait For Interrupt request.
  - PWR_STOPENTRY_WFE enter Stop mode with Wait For Event request.

Return values

- **None.:**

Notes

- In Stop 2 mode, all clocks in the VCORE domain are stopped. The PLL, MSIS, MSIK, HSI16 and HSE oscillators are disabled.
- Stop mode achieves the lowest power consumption while retaining the content of SRAM and registers. All clocks in the VCORE domain are stopped. The PLL, the MSI (MSIS and MSIK) RC, the HSI16 RC and the HSE crystal oscillators are disabled. The LSE or LSI is still running.
- The system clock when exiting from Stop mode can be either MSIS up to 24 MHz or HSI16, depending on software configuration.

HAL_PWREx_EnterSTOP3Mode

Function name

void HAL_PWREx_EnterSTOP3Mode (uint8_t STOPEntry)

Function description

Enter the whole system to Stop 3 mode.

Parameters

- **STOPEntry:** : Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values :
  - PWR_STOPENTRY_WFI enter Stop mode with Wait For Interrupt request.
  - PWR_STOPENTRY_WFE enter Stop mode with Wait For Event request.

Return values

- **None.:**

Notes

- Stop 3 is the lowest power mode with full retention, but the functional peripherals and sources of wakeup are reduced to the same ones than in Standby mode.
- Stop mode achieves the lowest power consumption while retaining the content of SRAM and registers. All clocks in the VCORE domain are stopped. The PLL, the MSI (MSIS and MSIK) RC, the HSI16 RC and the HSE crystal oscillators are disabled. The LSE or LSI is still running.
- The system clock when exiting from Stop mode can be either MSIS up to 24 MHz or HSI16, depending on software configuration.

HAL_PWREx_EnterSHUTDOWNMode

Function name

void HAL_PWREx_EnterSHUTDOWNMode (void)

Function description

Enter the whole system to Shutdown mode.

Return values

- **None.:**

Notes

- Shutdown mode allows the lowest power consumption. The internal regulator is switched off so that the VCORE domain is powered off. The PLL, the HSI16, the MSI (MSIS and MSIK), the LSI and the HSE oscillators are also switched off.
- The SRAMs and register contents are lost except for registers in the Backup domain. The BOR is not available in Shutdown mode. No power voltage monitoring is possible in this mode, therefore the switch to Backup domain is not supported.

HAL_PWREx_ConfigSRDDomain

Function name

void HAL_PWREx_ConfigSRDDomain (uint32_t SRDState)

Function description

Configure the SRD domain when the System in Stop 0/1/2 mode.

Parameters

- **SRDState:** : Specifies the SRD state. This parameter can be one of the following values :
  – PWR_SRD_DOMAIN_STOP : SRD in Stop mode when system goes to Stop 0/1/2 mode
  – PWR_SRD_DOMAIN_RUN : SRD in Run mode when system goes to Stop 0/1/2 mode

Return values

- **None.:**

HAL_PWREx_EnableUltraLowPowerMode

Function name

void HAL_PWREx_EnableUltraLowPowerMode (void)

Function description

Enable BOR ultra-low power mode.

Return values

- **None.:**

Notes

- BOR operating can be in discontinuous (ultra-low power) mode in Stop 1, Stop 2, Stop 3 and Standby modes and when the regulator is in Range 4 (Run, Sleep or Stop 0 mode).

HAL_PWREx_DisableUltraLowPowerMode

Function name

void HAL_PWREx_DisableUltraLowPowerMode (void)

Function description

Disable BOR ultra-low power mode.

Return values

- **None.:**

HAL_PWREx_S3WU_IRQHandler

Function name

void HAL_PWREx_S3WU_IRQHandler (uint32_t WakeUpPin)

Function description

This function handles the PWR Wake up from Stop 3 interrupt request.

Parameters

- **WakeUpPin:** : Specifies the wakeup pin interrupt to be handled. This parameter can be a combination of PWR Wake Up Pins (Default polarity and default pin selection).

Return values

- **None.:**

Notes

- This API should be called under the PWR_S3WU_IRQHandler().

HAL_PWREx_S3WUCallback

Function name

void HAL_PWREx_S3WUCallback (uint32_t WakeUpPin)

Function description

PWR S3WU interrupt callback.

Parameters

- **WakeUpPin:** : Specifies the wakeup pin interrupt to be handled. This parameter can be a combination of PWR Wake Up Pins (Default polarity and default pin selection).

Return values

- **None.:**

HAL_PWREx_EnableBatteryCharging

Function name

void HAL_PWREx_EnableBatteryCharging (uint32_t ResistorValue)

Function description

Enable the Battery charging.

Parameters

- **ResistorValue:** : Specifies the charging resistor. This parameter can be one of the following values :
  – PWR_BATTERY_CHARGING_RESISTOR_5 : 5 KOhm resistor.
  – PWR_BATTERY_CHARGING_RESISTOR_1_5 : 1.5 KOhm resistor.

Return values

- **None.:**

Notes

- When VDD is present, charge the external battery through an internal resistor.

HAL_PWREx_DisableBatteryCharging

Function name

void HAL_PWREx_DisableBatteryCharging (void)

Function description

Disable the Battery charging.

Return values

- **None.:**

HAL_PWREx_EnableVddUSB

Function name

void HAL_PWREx_EnableVddUSB (void)

Function description

Enable VDDUSB supply.

Return values

- **None.:**

Notes

- Remove VDDUSB electrical and logical isolation, once VDDUSB supply is present for consumption saving.

HAL_PWREx_DisableVddUSB

Function name

void HAL_PWREx_DisableVddUSB (void)

Function description

Disable VDDUSB supply.

Return values

- **None.:**

HAL_PWREx_EnableVddIO2

Function name

void HAL_PWREx_EnableVddIO2 (void)

Function description

Enable VDDIO2 supply.

Return values

- **None.:**

Notes

- Remove VDDIO2 electrical and logical isolation, once VDDIO2 supply is present for consumption saving.

HAL_PWREx_DisableVddIO2

Function name

void HAL_PWREx_DisableVddIO2 (void)

Function description

Disable VDDIO2 supply.

Return values

- **None.:**

HAL_PWREx_EnableVddA

Function name

void HAL_PWREx_EnableVddA (void)

Function description

Enable VDDA supply.

Return values

- **None.:**

Notes

- Remove VDDA electrical and logical isolation, once VDDA supply is present for consumption saving.

HAL_PWREx_DisableVddA

Function name

void HAL_PWREx_DisableVddA (void)

Function description

Disable VDDA supply.

Return values

- **None.:**

HAL_PWREx_EnableUVM

Function name

void HAL_PWREx_EnableUVM (void)

Function description

Enable the UVM Voltage Monitoring : VDDUSB versus 1.2 V.

Return values

- **None.:**

HAL_PWREx_DisableUVM

Function name

void HAL_PWREx_DisableUVM (void)

Function description

Disable the UVM Voltage Monitoring : VDDUSB versus 1.2 V.

Return values

- **None.:**

HAL_PWREx_EnableIO2VM

Function name

void HAL_PWREx_EnableIO2VM (void)

Function description

Enable the IO2VM Voltage Monitoring : VDDIO2 versus 0.9 V.

Return values

- **None.:**

HAL_PWREx_DisableIO2VM

Function name

void HAL_PWREx_DisableIO2VM (void)

Function description

Disable the IO2VM Voltage Monitoring : VDDIO2 versus 0.9 V.

Return values

- **None.:**

HAL_PWREx_EnableAVM1

Function name

void HAL_PWREx_EnableAVM1 (void)

Function description

Enable the AVM1 Voltage Monitoring : VDDA versus 1.6 V.

Return values

- **None.:**

HAL_PWREx_DisableAVM1

Function name

void HAL_PWREx_DisableAVM1 (void)

Function description

Disable the AVM1 Voltage Monitoring : VDDA versus 1.6 V.

Return values

- **None.:**

HAL_PWREx_EnableAVM2

Function name

void HAL_PWREx_EnableAVM2 (void)

Function description

Enable the AVM2 Voltage Monitoring : VDDA versus 1.8 V.

Return values

- **None.:**

HAL_PWREx_DisableAVM2

Function name

void HAL_PWREx_DisableAVM2 (void)

Function description

Disable the AVM2 Voltage Monitoring : VDDA versus 1.8 V.

Return values

- **None.:**

HAL_PWREx_EnableUSBHSTranceiverSupply

Function name

HAL_StatusTypeDef HAL_PWREx_EnableUSBHSTranceiverSupply (void)

Function description

Enable the internal USB HS transceiver supply.

Return values

- **HAL:** status.

HAL_PWREx_DisableUSBHSTranceiverSupply

Function name

void HAL_PWREx_DisableUSBHSTranceiverSupply (void)

Function description

Disable the internal USB HS transceiver supply.

Return values

- **HAL:** status.

HAL_PWREx_EnableOTGHSPHYLowPowerRetention

Function name

void HAL_PWREx_EnableOTGHSPHYLowPowerRetention (void)

Function description

Enable OTG_HS PHY power during low power modes (Stop2, Stop3 and Standby).

Return values

- **None.:**

HAL_PWREx_DisableOTGHSPHYLowPowerRetention

Function name

void HAL_PWREx_DisableOTGHSPHYLowPowerRetention (void)

Function description

Disable OTG_HS PHY power during low power modes (Stop2, Stop3 and Standby).

Return values

- **None.:**

HAL_PWREx_EnableVDD11USB

Function name

void HAL_PWREx_EnableVDD11USB (void)

Function description

Enable the VDD11USB.

Return values

- **None.:**

HAL_PWREx_DisableVDD11USB

Function name

void HAL_PWREx_DisableVDD11USB (void)

Function description

Disable the VDD11USB.

Return values

- **None.:**

HAL_PWREx_ConfigPVM

Function name

HAL_StatusTypeDef HAL_PWREx_ConfigPVM (PWR_PVMTypeDef * pConfigPVM)

Function description

Configure the voltage monitor threshold detected by the Peripheral voltage monitoring (PVM).

Parameters

- **pConfigPVM:** : Pointer to a PWR_PVMTypeDef structure that contains the PVM configuration information (PVMType and EventMode).

Return values

- **HAL:** Status.

HAL_PWREx_EnableMonitoring

Function name

void HAL_PWREx_EnableMonitoring (void)

Function description

Enable the VBAT and temperature monitoring.

Return values

- **None.:**

HAL_PWREx_DisableMonitoring

Function name

void HAL_PWREx_DisableMonitoring (void)

Function description

Disable the VBAT and temperature monitoring.

Return values

- **None.:**

HAL_PWREx_EnableUCPDStandbyMode

Function name

void HAL_PWREx_EnableUCPDStandbyMode (void)

Function description

Enable UCPD configuration memorization in Standby mode.

Return values

- **None.:**

HAL_PWREx_DisableUCPDStandbyMode

Function name

void HAL_PWREx_DisableUCPDStandbyMode (void)

Function description

Disable UCPD configuration memorization in Standby mode.

Return values

- **None.:**

Notes

- This function must be called on exiting the Standby mode and before any UCPD configuration update.

HAL_PWREx_EnableUCPDDeadBattery

Function name

void HAL_PWREx_EnableUCPDDeadBattery (void)

Function description

Enable dead battery behavior.

Return values

- **None.:**

Notes

- After exiting reset, the USB Type-C (dead battery) behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to handover control to the UCPD (the UCPD must be initialized before doing the disable).

HAL_PWREx_DisableUCPDDeadBattery

Function name

void HAL_PWREx_DisableUCPDDeadBattery (void)

Function description

Disable dead battery behavior.

Return values

- **None.:**

Notes

- After exiting reset, the USB Type-C (dead battery) behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to handover control to the UCPD (the UCPD must be initialized before doing the disable).

HAL_PWREx_PVD_PVM_IRQHandler

Function name

void HAL_PWREx_PVD_PVM_IRQHandler (void)

Function description

This function handles the PWR PVD/PVM interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the PVD_PVM_IRQHandler().

HAL_PWREx_UVMCallback

Function name

void HAL_PWREx_UVMCallback (void)

Function description

PWR UVM interrupt callback.

Return values

- **None.:**

HAL_PWREx_IO2VMCallback

Function name

void HAL_PWREx_IO2VMCallback (void)

Function description

PWR IO2VM interrupt callback.

Return values

- **None.:**

HAL_PWREx_AVM1Callback

Function name

void HAL_PWREx_AVM1Callback (void)

Function description

PWR AVM1 interrupt callback.

Return values

- **None.:**

HAL_PWREx_AVM2Callback

Function name

void HAL_PWREx_AVM2Callback (void)

Function description

PWR AVM2 interrupt callback.

Return values

- **None.:**

HAL_PWREx_EnableSRAM2ContentStandbyRetention

Function name

void HAL_PWREx_EnableSRAM2ContentStandbyRetention (uint32_t SRAM2Pages)

Function description

Enable SRAM2 page(s) content retention in Stop 3 and Standby mode.

Parameters

- **SRAM2Pages:** : Specifies the SRAM2 pages. This parameter can be one of the following values :
  – PWR_SRAM2_PAGE1_STANDBY : SRAM2 page 1 retention.
  – PWR_SRAM2_PAGE2_STANDBY : SRAM2 page 2 retention.
  – PWR_SRAM2_FULL_STANDBY : SRAM2 page 1 and page 2 retention.

Return values

- **None.:**

Notes

- When RRSx bit is set, SRAM2 is powered by the low-power regulator in Stop 3 and Standby mode and its content is kept.

HAL_PWREx_DisableSRAM2ContentStandbyRetention

Function name

void HAL_PWREx_DisableSRAM2ContentStandbyRetention (uint32_t SRAM2Pages)

Function description

Disable SRAM2 page(s) content retention in Stop 3 and Standby mode.

Parameters

- **SRAM2Pages:** : Specifies the SRAM2 pages. This parameter can be one of the following values :
  – PWR_SRAM2_PAGE1_STANDBY : SRAM2 page 1 retention.
  – PWR_SRAM2_PAGE2_STANDBY : SRAM2 page 2 retention.
  – PWR_SRAM2_FULL_STANDBY : SRAM2 page 1 and page 2 retention.

Return values

- **None.:**

Notes

- When RRSx bit is reset, SRAM2 is powered off in Stop 3 and Standby mode and its content is lost.

HAL_PWREx_EnableRAMsContentStopRetention

Function name

void HAL_PWREx_EnableRAMsContentStopRetention (uint32_t RAMSelection)

Function description

Enable RAM page(s) content retention in Stop mode (Stop 0, 1, 2, 3).

Parameters

- **RAMSelection:** Specifies RAM page(s) to be retained in Stop mode. This parameter can be one or a combination of the same memory PWR Extended RAM Contents Stop Retention.

Return values

- **None.:**

Notes

- When enabling content retention for a given RAM, memory kept powered in Stop mode. (Not optimized power consumption)

HAL_PWREx_DisableRAMsContentStopRetention

Function name

void HAL_PWREx_DisableRAMsContentStopRetention (uint32_t RAMSelection)

Function description

Disable RAM page(s) content lost in Stop mode (Stop 0, 1, 2, 3).

Parameters

- **RAMSelection:** Specifies RAM page(s) to be lost in Stop mode. This parameter can be one or a combination of the same memory PWR Extended RAM Contents Stop Retention.

Return values

- **None.:**

Notes

- When disabling content lost for a given RAM, memory powered down in Stop mode. (Optimized power consumption)

HAL_PWREx_EnableRAMsContentRunRetention

Function name

void HAL_PWREx_EnableRAMsContentRunRetention (uint32_t RAMSelection)

Function description

Enable RAMs full content retention in Run mode.

Parameters

- **RAMSelection:** : Specifies the SRAM content to be retained in Run mode. This parameter can be one or a combination of the following values :
  – PWR_SRAM1_FULL_RUN : SRAM1 full content retention.
  – PWR_SRAM2_FULL_RUN : SRAM2 full content retention.
  – PWR_SRAM3_FULL_RUN : SRAM3 full content retention (available only for STM32U575xx, STM32U585xx STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices).
  – PWR_SRAM4_FULL_RUN : SRAM4 full content retention.
  – PWR_SRAM5_FULL_RUN : SRAM5 full content retention (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices).
  – PWR_SRAM6_FULL_RUN : SRAM6 full content retention (available only for STM32U5Fxxx and STM32U5Gxxx devices).

Return values

- **None.:**

HAL_PWREx_DisableRAMsContentRunRetention

Function name

void HAL_PWREx_DisableRAMsContentRunRetention (uint32_t RAMSelection)

Function description

Disable RAMs full content retention in Run mode.

Parameters

- **RAMSelection:** : Specifies the SRAM content to be lost in Run mode. This parameter can be one or a combination of the following values :
  - PWR_SRAM1_FULL_RUN : SRAM1 full content lost.
  - PWR_SRAM2_FULL_RUN : SRAM2 full content lost.
  - PWR_SRAM3_FULL_RUN : SRAM3 full content lost (available only for STM32U575xx, STM32U585xx STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices).
  - PWR_SRAM4_FULL_RUN : SRAM4 full content lost.
  - PWR_SRAM5_FULL_RUN : SRAM5 full content retention (available only for STM32U59xxx, STM32U5Axxx, STM32U5Fxxx and STM32U5Gxxx devices).
  - PWR_SRAM6_FULL_RUN : SRAM6 full content retention (available only for STM32U5Fxxx and STM32U5Gxxx devices).

Return values

- **None.:**

HAL_PWREx_EnableFlashFastWakeUp

Function name

void HAL_PWREx_EnableFlashFastWakeUp (void)

Function description

Enable the flash memory fast wakeup from Stop 0 and Stop 1 modes.

Return values

- **None.:**

Notes

- This feature is used to obtain the best trade-off between low-power consumption and wakeup time when exiting the Stop 0 or Stop 1 modes. When this feature is enabled, the Flash memory remains in normal mode in Stop 0 and Stop 1 modes, which offers a faster startup time with higher consumption.

HAL_PWREx_DisableFlashFastWakeUp

Function name

void HAL_PWREx_DisableFlashFastWakeUp (void)

Function description

Disable the Flash Power Down in Stop mode.

Return values

- **None.:**

Notes

- This feature is used to obtain the best trade-off between low-power consumption and wakeup time when exiting the Stop 0 or Stop 1 modes. When this feature is disabled, the Flash memory enters low-power mode in Stop 0 and Stop 1 modes, which causes a slower startup time with lower consumption.

HAL_PWREx_EnableSRAM4FastWakeUp

Function name

void HAL_PWREx_EnableSRAM4FastWakeUp (void)

Function description

Enable the SRAM4 memory fast wakeup from Stop 0, Stop 1 and Stop 2 modes.

Return values

- **None.:**

Notes

- This bit is used to obtain the best trade-off between low-power consumption and wakeup time. SRAM4 wakeup time increases the wakeup time when exiting Stop 0, Stop 1 and Stop 2 modes, and also increases the LPDMA access time to SRAM4 during Stop modes.

HAL_PWREx_DisableSRAM4FastWakeUp

Function name

void HAL_PWREx_DisableSRAM4FastWakeUp (void)

Function description

Disable the SRAM4 memory fast wakeup from Stop 0, Stop 1 and Stop 2 modes.

Return values

- **None.:**

Notes

- This bit is used to obtain the best trade-off between low-power consumption and wakeup time. SRAM4 wakeup time increases the wakeup time when exiting Stop 0, Stop 1 and Stop 2 modes, and also increases the LPDMA access time to SRAM4 during Stop modes.

HAL_PWREx_EnableBkupRAMRetention

Function name

HAL_StatusTypeDef HAL_PWREx_EnableBkupRAMRetention (void)

Function description

Enable the Backup RAM retention in Standby, Shutdown and VBAT modes.

Return values

- **HAL:** Status.

Notes

- If BREN is reset, the backup RAM can still be used in Run, Sleep and Stop modes. However, its content is lost in Standby, Shutdown and VBAT modes.
- This bit can be enabled only when LDO regulator is selected as source supply.

HAL_PWREx_DisableBkupRAMRetention

Function name

void HAL_PWREx_DisableBkupRAMRetention (void)

Function description

Disable the Backup RAM retention in Standby, Shutdown and VBAT modes.

Return values

- **None.:**

Notes

- If BREN is reset, the backup RAM can still be used in Run, Sleep and Stop modes. However, its content is lost in Standby, Shutdown and VBAT modes. This bit can be writte

HAL_PWREx_EnablePullUpPullDownConfig

Function name

void HAL_PWREx_EnablePullUpPullDownConfig (void)

Function description

Enable pull-up and pull-down configuration.

Return values

- **None.:**

Notes

- When APC bit is set, the I/O pull-up and pull-down configurations defined in PWR_PUCRx and PWR_PDCRx registers are applied in Standby and Shutdown modes.
- Pull-up set by PUy bit of PWR_PUCRx register is not activated if the corresponding PDy bit of PWR_PDCRx register is also set (pull-down configuration priority is higher). HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() API's ensure there is no conflict when setting PUy or PDy bit.

HAL_PWREx_DisablePullUpPullDownConfig

Function name

void HAL_PWREx_DisablePullUpPullDownConfig (void)

Function description

Disable pull-up and pull-down configuration.

Return values

- **None.:**

Notes

- When APC bit is cleared, the I/O pull-up and pull-down configurations defined in PWR_PUCRx and PWR_PDCRx registers are not applied in Standby and Shutdown modes.

HAL_PWREx_EnableGPIOPullUp

Function name

HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullUp (uint32_t GPIO_Port, uint32_t GPIO_Pin)

Function description

Enable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO_Port:** : Specify the IO port. This parameter can be a value of PWR Extended GPIO Port.
- **GPIO_Pin:** : Specify the I/O pins numbers. This parameter can be a value of PWR Extended GPIO Pin Mask.

Return values

- **HAL:** Status.

Notes

- Set the relevant PUy bits of PWR_PUCRx register to configure the I/O in pull-up state in Standby and Shutdown modes.
- This state is effective in Standby and Shutdown modes only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.
- The configuration is lost when exiting the Shutdown mode due to the power-on reset, maintained when exiting the Standby mode.
- To avoid any conflict at Standby and Shutdown modes exits, the corresponding PDy bit of PWR_PDCRx register is cleared unless it is reserved.
- Even if a PUy bit to set is reserved, the other PUy bits entered as input parameter at the same time are set.

HAL_PWREx_DisableGPIOPullUp

Function name

HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullUp (uint32_t GPIO_Port, uint32_t GPIO_Pin)

Function description

Disable GPIO pull-up state in Standby mode and Shutdown modes.

Parameters

- **GPIO_Port:** : Specify the IO port. This parameter can be a value of PWR Extended GPIO Port.
- **GPIO_Pin:** : Specify the I/O pins numbers. This parameter can be a value of PWR Extended GPIO Pin Mask.

Return values

- **HAL:** Status.

Notes

- Reset the relevant PUy bits of PWR_PUCRx register used to configure the I/O in pull-up state in Standby and Shutdown modes.
- Even if a PUy bit to reset is reserved, the other PUy bits entered as input parameter at the same time are reset.

HAL_PWREx_EnableGPIOPullDown

Function name

HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullDown (uint32_t GPIO_Port, uint32_t GPIO_Pin)

Function description

Enable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO_Port:** : Specify the IO port. This parameter can be a value of PWR Extended GPIO Port.
- **GPIO_Pin:** : Specify the I/O pins numbers. This parameter can be a value of PWR Extended GPIO Pin Mask.

Return values

- **HAL:** Status.

Notes

- Set the relevant PDy bits of PWR_PDCRx register to configure the I/O in pull-down state in Standby and Shutdown modes.
- This state is effective in Standby and Shutdown modes only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.
- The configuration is lost when exiting the Shutdown mode due to the power-on reset, maintained when exiting the Standby mode.
- To avoid any conflict at Standby and Shutdown modes exits, the corresponding PUy bit of PWR_PUCRx register is cleared unless it is reserved.
- Even if a PDy bit to set is reserved, the other PDy bits entered as input parameter at the same time are set.

HAL_PWREx_DisableGPIOPullDown

Function name

HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullDown (uint32_t GPIO_Port, uint32_t GPIO_Pin)

Function description

Disable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO_Port:** : Specify the IO port. This parameter can be a value of PWR Extended GPIO Port.
- **GPIO_Pin:** : Specify the I/O pins numbers. This parameter can be a value of PWR Extended GPIO Pin Mask.

Return values

- **HAL:** Status.

Notes

- Reset the relevant PDy bits of PWR_PDCRx register used to configure the I/O in pull-down state in Standby and Shutdown modes.
- Even if a PDy bit to reset is reserved, the other PDy bits entered as input parameter at the same time are reset.

62.3 PWREx Firmware driver defines

The following section lists the various define and macros of the module.

62.3.1 PWREx

PWREx

PWR Extended Exported Macros

__HAL_PWR_UVM_EXTI_ENABLE_IT

Description:

- Enable the UVM Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_UVM_EXTI_DISABLE_IT

Description:

- Disable the UVM Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_UVM_EXTI_ENABLE_EVENT

Description:
- Enable the UVM Event Line.

Return value:
- None.

__HAL_PWR_UVM_EXTI_DISABLE_EVENT

Description:
- Disable the UVM Event Line.

Return value:
- None.

__HAL_PWR_UVM_EXTI_ENABLE_RISING_EDGE

Description:
- Enable the UVM Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_UVM_EXTI_DISABLE_RISING_EDGE

Description:
- Disable the UVM Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_UVM_EXTI_ENABLE_FALLING_EDGE

Description:
- Enable the UVM Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_UVM_EXTI_DISABLE_FALLING_EDGE

Description:
- Disable the UVM Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_UVM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:
- Enable the UVM Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_UVM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:
- Disable the UVM Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_UVM_EXTI_GENERATE_SWIT

Description:

- Generate a Software Interrupt on UVM EXTI Line.

Return value:

- None

__HAL_PWR_UVM_EXTI_GET_FLAG

Description:

- Check whether the specified UVM EXTI flag is set or not.

Return value:

- EXTI: UVM Line Status.

__HAL_PWR_UVM_EXTI_CLEAR_FLAG

Description:

- Clear the UVM EXTI flag.

Return value:

- None.

__HAL_PWR_IO2VM_EXTI_ENABLE_IT

Description:

- Enable the IO2VM Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_IO2VM_EXTI_DISABLE_IT

Description:

- Disable the IO2VM Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_IO2VM_EXTI_ENABLE_EVENT

Description:

- Enable the IO2VM Event Line.

Return value:

- None.

__HAL_PWR_IO2VM_EXTI_DISABLE_EVENT

Description:

- Disable the IO2VM Event Line.

Return value:

- None.

__HAL_PWR_IO2VM_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the IO2VM Extended Interrupt Rising Trigger.

Return value:

- None.

__HAL_PWR_IO2VM_EXTI_DISABLE_RISING_EDGE

Description:
- Disable the IO2VM Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_IO2VM_EXTI_ENABLE_FALLING_EDGE

Description:
- Enable the IO2VM Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_IO2VM_EXTI_DISABLE_FALLING_EDGE

Description:
- Disable the IO2VM Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_IO2VM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:
- Enable the IO2VM Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_IO2VM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:
- Disable the IO2VM Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_IO2VM_EXTI_GENERATE_SWIT

Description:
- Generate a Software Interrupt on IO2VM EXTI Line.

Return value:
- None

__HAL_PWR_IO2VM_EXTI_GET_FLAG

Description:
- Check whether the specified IO2VM EXTI flag is set or not.

Return value:
- EXTI: IO2VM Line Status.

__HAL_PWR_IO2VM_EXTI_CLEAR_FLAG

Description:
- Clear the IO2VM EXTI flag.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_ENABLE_IT

Description:
- Enable the AVM1 Extended Interrupt Line.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_DISABLE_IT

Description:
- Disable the AVM1 Extended Interrupt Line.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_ENABLE_EVENT

Description:
- Enable the AVM1 Event Line.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_DISABLE_EVENT

Description:
- Disable the AVM1 Event Line.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_ENABLE_RISING_EDGE

Description:
- Enable the AVM1 Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_DISABLE_RISING_EDGE

Description:
- Disable the AVM1 Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_ENABLE_FALLING_EDGE

Description:
- Enable the AVM1 Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_DISABLE_FALLING_EDGE

Description:
- Disable the AVM1 Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_AVM1_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the AVM1 Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_AVM1_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the AVM1 Extended Interrupt Rising & Falling Trigger.

Return value:

- None.

__HAL_PWR_AVM1_EXTI_GENERATE_SWIT

Description:

- Generate a Software Interrupt on AVM1 EXTI Line.

Return value:

- None

__HAL_PWR_AVM1_EXTI_GET_FLAG

Description:

- Check whether the specified AVM1 EXTI flag is set or not.

Return value:

- EXTI: AVM1 Line Status.

__HAL_PWR_AVM1_EXTI_CLEAR_FLAG

Description:

- Clear the AVM1 EXTI flag.

Return value:

- None.

__HAL_PWR_AVM2_EXTI_ENABLE_IT

Description:

- Enable the AVM2 Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_AVM2_EXTI_DISABLE_IT

Description:

- Disable the AVM2 Extended Interrupt Line.

Return value:

- None.

__HAL_PWR_AVM2_EXTI_ENABLE_EVENT

Description:

- Enable the AVM2 Event Line.

Return value:

- None.

__HAL_PWR_AVM2_EXTI_DISABLE_EVENT

Description:
- Disable the AVM2 Event Line.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_ENABLE_RISING_EDGE

Description:
- Enable the AVM2 Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_DISABLE_RISING_EDGE

Description:
- Disable the AVM2 Extended Interrupt Rising Trigger.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_ENABLE_FALLING_EDGE

Description:
- Enable the AVM2 Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_DISABLE_FALLING_EDGE

Description:
- Disable the AVM2 Extended Interrupt Falling Trigger.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:
- Enable the AVM2 Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:
- Disable the AVM2 Extended Interrupt Rising & Falling Trigger.

Return value:
- None.

__HAL_PWR_AVM2_EXTI_GENERATE_SWIT

Description:
- Generate a Software Interrupt on AVM2 EXTI Line.

Return value:
- None

__HAL_PWR_AVM2_EXTI_GET_FLAG

Description:

- Check whether the specified AVM2 EXTI flag is set or not.

Return value:

- EXTI: AVM2 Line Status.

__HAL_PWR_AVM2_EXTI_CLEAR_FLAG

Description:

- Clear the AVM2 EXTI flag.

Return value:

- None.

__HAL_PWR_VOLTAGESCALING_CONFIG

Description:

- Configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__: : Specifies the regulator output voltage to achieve a tradeoff between performance and power consumption. This parameter can be one of the following values :
  - PWR_REGULATOR_VOLTAGE_SCALE1 : Regulator voltage output scale 1. Provides a typical output voltage at 1.2 V. Used when system clock frequency is up to 160 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE2 : Regulator voltage output scale 2. Provides a typical output voltage at 1.1 V. Used when system clock frequency is up to 100 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE3 : Regulator voltage output scale 3. Provides a typical output voltage at 1.0 V. Used when system clock frequency is up to 50 MHz.
  - PWR_REGULATOR_VOLTAGE_SCALE4 : Regulator voltage output scale 4. Provides a typical output voltage at 0.9 V. Used when system clock frequency is up to 24 MHz.

Return value:

- None.

Notes:

- This macro is similar to HAL_PWREx_ControlVoltageScaling() API but doesn't check whether or not VOSREADY flag is set. User may resort to __HAL_PWR_GET_FLAG() macro to check VOSF bit state.

PWR Extended GPIO Pin Mask

PWR_GPIO_BIT_0

GPIO port I/O pin 0

PWR_GPIO_BIT_1

GPIO port I/O pin 1

PWR_GPIO_BIT_2

GPIO port I/O pin 2

PWR_GPIO_BIT_3

GPIO port I/O pin 3

PWR_GPIO_BIT_4

GPIO port I/O pin 4

PWR_GPIO_BIT_5

GPIO port I/O pin 5

PWR_GPIO_BIT_6

GPIO port I/O pin 6

PWR_GPIO_BIT_7
GPIO port I/O pin 7

PWR_GPIO_BIT_8
GPIO port I/O pin 8

PWR_GPIO_BIT_9
GPIO port I/O pin 9

PWR_GPIO_BIT_10
GPIO port I/O pin 10

PWR_GPIO_BIT_11
GPIO port I/O pin 11

PWR_GPIO_BIT_12
GPIO port I/O pin 12

PWR_GPIO_BIT_13
GPIO port I/O pin 13

PWR_GPIO_BIT_14
GPIO port I/O pin 14

PWR_GPIO_BIT_15
GPIO port I/O pin 15

PWR Extended GPIO Port

PWR_GPIO_A
GPIO port A

PWR_GPIO_B
GPIO port B

PWR_GPIO_C
GPIO port C

PWR_GPIO_D
GPIO port D

PWR_GPIO_E
GPIO port E

PWR_GPIO_F
GPIO port F

PWR_GPIO_G
GPIO port G

PWR_GPIO_H
GPIO port H

PWR_GPIO_I
GPIO port I

PWR_GPIO_J
GPIO port J

PWR Extended PVM Interrupt and Event Mode

PWR_PVM_MODE_NORMAL

Basic Mode is used

PWR_PVM_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

PWR_PVM_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

PWR_PVM_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

PWR_PVM_MODE_EVENT_RISING

Event Mode with Rising edge trigger detection

PWR_PVM_MODE_EVENT_FALLING

Event Mode with Falling edge trigger detection

PWR_PVM_MODE_EVENT_RISING_FALLING

Event Mode with Rising/Falling edge trigger detection

PWR Extended Voltage Monitoring Type

PWR_UVM

Independent USB voltage monitor (VDDUSB versus 1.2 V)

PWR_IO2VM

Independent VDDIO2 voltage monitor (VDDIO2 versus 0.9 V)

PWR_AVM1

Independent VDDA voltage monitor (VDDA versus 1.6 V)

PWR_AVM2

Independent VDDA voltage monitor (VDDA versus 1.8 V)

PWR Extended RAM Contents Stop Retention

PWR_SRAM1_PAGE1_STOP

SRAM1 page 1 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE2_STOP

SRAM1 page 2 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE3_STOP

SRAM1 page 3 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE4_STOP

SRAM1 page 4 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE5_STOP

SRAM1 page 5 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE6_STOP

SRAM1 page 6 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE7_STOP

SRAM1 page 7 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE8_STOP

SRAM1 page 8 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE9_STOP

SRAM1 page 9 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE10_STOP

SRAM1 page 10 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE11_STOP

SRAM1 page 11 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_PAGE12_STOP

SRAM1 page 12 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM1_FULL_STOP

SRAM1 all pages retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM2_PAGE1_STOP

SRAM2 page 1 (8 KB) retention in Stop modes (Stop 0, 1, 2)

PWR_SRAM2_PAGE2_STOP

SRAM2 page 2 (54 KB) retention in Stop modes (Stop 0, 1, 2)

PWR_SRAM2_FULL_STOP

SRAM2 all pages retention in Stop modes (Stop 0, 1, 2)

PWR_SRAM3_PAGE1_STOP

SRAM3 page 1 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE2_STOP

SRAM3 page 2 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE3_STOP

SRAM3 page 3 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE4_STOP

SRAM3 page 4 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE5_STOP

SRAM3 page 5 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE6_STOP

SRAM3 page 6 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE7_STOP

SRAM3 page 7 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE8_STOP

SRAM3 page 8 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE9_STOP

SRAM3 page 9 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE10_STOP
SRAM3 page 10 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE11_STOP
SRAM3 page 11 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE12_STOP
SRAM3 page 12 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_PAGE13_STOP
SRAM3 page 13 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM3_FULL_STOP
SRAM3 all pages retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM4_FULL_STOP
SRAM4 retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE1_STOP
SRAM5 page 1 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE2_STOP
SRAM5 page 2 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE3_STOP
SRAM5 page 3 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE4_STOP
SRAM5 page 4 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE5_STOP
SRAM5 page 5 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE6_STOP
SRAM5 page 6 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE7_STOP
SRAM5 page 7 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE8_STOP
SRAM5 page 8 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE9_STOP
SRAM5 page 9 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE10_STOP
SRAM5 page 10 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE11_STOP
SRAM5 page 11 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE12_STOP
SRAM5 page 12 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_PAGE13_STOP
SRAM5 page 13 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM5_FULL_STOP

SRAM5 all pages retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE1_STOP

SRAM6 page 1 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE2_STOP

SRAM6 page 2 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE3_STOP

SRAM6 page 3 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE4_STOP

SRAM6 page 4 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE5_STOP

SRAM6 page 5 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE6_STOP

SRAM6 page 6 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE7_STOP

SRAM6 page 7 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_PAGE8_STOP

SRAM6 page 8 (64 KB) retention in Stop modes (Stop 0, 1, 2, 3)

PWR_SRAM6_FULL_STOP

SRAM6 all pages retention in Stop modes (Stop 0, 1, 2, 3)

PWR_ICACHE_FULL_STOP

ICACHE page retention in Stop modes (Stop 0, 1, 2, 3)

PWR_DCACHE1_FULL_STOP

DCACHE1 page retention in Stop modes (Stop 0, 1, 2, 3)

PWR_DCACHE2_FULL_STOP

DCACHE2 page retention in Stop modes (Stop 0, 1, 2, 3)

PWR_DMA2DRAM_FULL_STOP

DMA2D RAM retention in Stop modes (Stop 0, 1, 2, 3)

PWR_PERIPHRAM_FULL_STOP

FMAC, FDCAN and USB RAM retention in Stop modes (Stop 0, 1, 2, 3)

PWR_PKA32RAM_FULL_STOP

PKA32 RAM retention in Stop modes (Stop 0, 1, 2, 3)

PWR_GRAPHICPRAM_FULL_STOP

LTDC, GFXMMU retention in Stop modes (Stop 0, 1, 2, 3)

PWR_DSIRAM_FULL_STOP

DSI RAM retention in Stop modes (Stop 0, 1, 2, 3)

PWR_JPEGRAM_FULL_STOP

JPEG RAM retention in Stop modes (Stop 0, 1, 2, 3)

PWR Extended Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE1
Voltage scaling range 1

PWR_REGULATOR_VOLTAGE_SCALE2
Voltage scaling range 2

PWR_REGULATOR_VOLTAGE_SCALE3
Voltage scaling range 3

PWR_REGULATOR_VOLTAGE_SCALE4
Voltage scaling range 4

PWR Extended SRAM2 Contents Standby Retention

PWR_SRAM2_PAGE1_STANDBY
SRAM2 page 1 (8 KB) retention in Stop 3 and Standby modes

PWR_SRAM2_PAGE2_STANDBY
SRAM2 page 2 (54 KB) retention in Stop 3 and Standby modes

PWR_SRAM2_FULL_STANDBY
SRAM2 all pages retention in Stop 3 and Standby modes

PWR Extended SRAM Contents Run Retention

PWR_SRAM1_FULL_RUN
SRAM1 full retention in Run mode

PWR_SRAM2_FULL_RUN
SRAM2 full retention in Run mode

PWR_SRAM3_FULL_RUN
SRAM3 full retention in Run mode

PWR_SRAM4_FULL_RUN
SRAM4 full retention in Run mode

PWR_SRAM5_FULL_RUN
SRAM5 full retention in Run mode

PWR_SRAM6_FULL_RUN
SRAM6 full retention in Run mode

PWREx SRD Domain State

PWR_SRD_DOMAIN_STOP
SRD in Stop mode when system goes to Stop 0/1/2 mode

PWR_SRD_DOMAIN_RUN
SRD in Run mode when system goes to Stop 0/1/2 mode

PWR Extended Supply Configuration

PWR_LDO_SUPPLY
LDO supply

PWR_SMPS_SUPPLY
SMPS supply

PWR Extended Battery Charging Resistor Selection

PWR_BATTERY_CHARGING_RESISTOR_5

VBAT charging through a 5 kOhms resistor

PWR_BATTERY_CHARGING_RESISTOR_1_5

VBAT charging through a 1.5 kOhms resistor

63 HAL RAMCFG Generic Driver

63.1 RAMCFG Firmware driver registers structures

63.1.1 __RAMCFG_HandleTypeDef

__RAMCFG_HandleTypeDef is defined in the stm32u5xx_hal_ramcfg.h

Data Fields

- ***RAMCFG_TypeDef * Instance***
- ***__IO HAL_RAMCFG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DetectSingleErrorCallback***
- ***void(* DetectDoubleErrorCallback***

Field Documentation

- ***RAMCFG_TypeDef* __RAMCFG_HandleTypeDef::Instance***
  RAMCFG Register Base Address
- ***__IO HAL_RAMCFG_StateTypeDef __RAMCFG_HandleTypeDef::State***
  RAMCFG State
- ***__IO uint32_t __RAMCFG_HandleTypeDef::ErrorCode***
  RAMCFG Error Code
- ***void(* __RAMCFG_HandleTypeDef::MspInitCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG MSP Init Callback
- ***void(* __RAMCFG_HandleTypeDef::MspDeInitCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG MSP DeInit Callback
- ***void(* __RAMCFG_HandleTypeDef::DetectSingleErrorCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG Single Error Detect Callback
- ***void(* __RAMCFG_HandleTypeDef::DetectDoubleErrorCallback)(struct __RAMCFG_HandleTypeDef *hramcfg)***
  RAMCFG Double Error Detect Callback

63.2 RAMCFG Firmware driver API description

The following section lists the various functions of the RAMCFG library.

63.2.1 RAMCFG Peripheral features

- Each SRAM is managed by a RAMCFG instance.
- Each SRAM can be erased independently through its RAMCFG instance.
- The wait state value for each SRAM can be configured independently through its RAMCFG instance.
- SRAM2 is divided to 64 pages with 1 kB granularity. Each page can be write protected independently through its RAMCFG instance.
- SRAM2, SRAM3 and BKPRAM support ECC correction feature. This mechanism adopts the Single Error Correction Double Error Detection (SECDED) algorithm. This feature provides the following information:
  – Single error address.
  – Double error address.
  ==

63.2.2 How to use this driver

1. Call HAL_RAMCFG_Init() to initialize the RAMCFG peripheral before using any feature. Call HAL_RAMCFG_DeInit() to de-initialize the RAMCFG when using this peripheral is no more needed or a hardware issue has occurred. (+) HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

(#) Call HAL_RAMCFG_Init() to initialize the RAMCFG peripheral before using any feature. Call HAL_RAMCFG_DeInit() to de-initialize the RAMCFG when using this peripheral is no more needed or a hardware issue has occurred.

- HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

ECC feature

- Call HAL_RAMCFG_StartECC() and HAL_RAMCFG_StopECC() to enable and disable ECC hardware mechanism.
  – When ECC feature is previously enabled (case of option byte activation), calling HAL_RAMCFG_StartECC() is recommended to enable the ECC address latching feature.
- Call HAL_RAMCFG_EnableNotification() and HAL_RAMCFG_DisableNotification() to enable and disable ECC interrupts. Interrupts can be:
  – Single error interrupt.
  – Double error interrupt.
  – Double error interrupt redirected to Non maskable interrupt (NMI).
- Call HAL_RAMCFG_GetSingleErrorAddress() to get the address of the last fail RAM word detected (only for single error) and call HAL_RAMCFG_GetDoubleErrorAddress() to get the address of the last fail RAM word detected (only for double error).
- Call HAL_RAMCFG_IsECCErrorDetected() to check if an ECC single/double error was detected. This API is used in silent mode (No ECC interrupt is enabled).

Wait State feature

- Call HAL_RAMCFG_ConfigWaitState() to configure the wait state value for a given SRAM according to the HCLK frequency and voltage range scaling.
- Call HAL_RAMCFG_GetWaitState() to get the current configured wait state value.

Write protection feature

- Call HAL_RAMCFG_EnableWriteProtection() to enable the write protection for the given SRAM2 page(s).
- There is no API to disable write protection as this feature can be disabled only by a global peripheral reset or system reset.
- Any write access to a write protected area of SRAM2 causes a HardFault interrupt.

Erase feature

- Call HAL_RAMCFG_Erase() to launch a hardware erase for the given SRAM.
- The erase value is equal to 0 when launching erase hardware through RAMCFG.
- SRAM2 write protected pages are erased when performing an erase through RAMCFG.

RAMCFG HAL driver macros list

Below the list of used macros in RAMCFG HAL driver.

- __HAL_RAMCFG_ENABLE_IT : Enable the specified RAMCFG interrupts.
- __HAL_RAMCFG_DISABLE_IT : Disable the specified RAMCFG interrupts.
- __HAL_RAMCFG_GET_FLAG : Get the RAMCFG pending flags.
- __HAL_RAMCFG_CLEAR_FLAG : Clear the RAMCFG pending flags.
- __HAL_RAMCFG_GET_IT_SOURCE : Check whether the specified RAMCFG interrupt source is enabled or not.

63.2.3 Initialization and de-initialization Functions

This section provides functions allowing to initialize and de-initialize the RAMCFG instance.

The HAL_RAMCFG_Init() function follows the RAMCFG instance configuration procedures as described in the reference manual. The HAL_RAMCFG_DeInit() function allows to deinitialize the RAMCFG instance. HAL_RAMCFG_Init() and HAL_RAMCFG_DeInit() APIs do not change the activation status of ECC feature. It is managed by HAL_RAMCFG_StartECC(), HAL_RAMCFG_StopECC() or option bytes (When available on the device).

This section contains the following APIs:

- *HAL_RAMCFG_Init()*
- *HAL_RAMCFG_DeInit()*
- *HAL_RAMCFG_MspInit()*
- *HAL_RAMCFG_MspDeInit()*

63.2.4 ECC Operations Functions

This section provides functions allowing to manage ECC feature provided by the RAMCFG peripheral.

The HAL_RAMCFG_StartECC() function allows starting the ECC mechanism and enabling ECC address latching for the selected RAMCFG instance. The HAL_RAMCFG_StopECC() function allows stopping the ECC mechanism and disabling ECC address latching for the selected RAMCFG instance. The HAL_RAMCFG_EnableNotification() function allows enabling interrupts for single ECC error, double ECC error and NMI error. The HAL_RAMCFG_DisableNotification() function allows disabling interrupts for single ECC error, double ECC error. When NMI interrupt is enabled it can only be disabled by a global peripheral reset or by a system reset. The HAL_RAMCFG_IsECCErrorDetected() function allows to check if an ECC error has occurred. The HAL_RAMCFG_GetSingleErrorAddress() function allows to get the address of the last single ECC error detected. The HAL_RAMCFG_GetDoubleErrorAddress() function allows to get the address of the last double ECC error detected.

This section contains the following APIs:

- *HAL_RAMCFG_StartECC()*
- *HAL_RAMCFG_StopECC()*
- *HAL_RAMCFG_EnableNotification()*
- *HAL_RAMCFG_DisableNotification()*
- *HAL_RAMCFG_IsECCSingleErrorDetected()*
- *HAL_RAMCFG_IsECCDoubleErrorDetected()*
- *HAL_RAMCFG_GetSingleErrorAddress()*
- *HAL_RAMCFG_GetDoubleErrorAddress()*

63.2.5 Wait State Functions

This section provides functions allowing to get and configure the wait state value.

The HAL_RAMCFG_ConfigWaitState() function allows configuring the wait state value. The HAL_RAMCFG_GetWaitState() function allows getting the current wait state value.

This section contains the following APIs:

- *HAL_RAMCFG_ConfigWaitState()*
- *HAL_RAMCFG_GetWaitState()*

63.2.6 Write Protection Functions

This section provides functions to enable write protection feature for the page(s) of SRAM2.

The HAL_RAMCFG_EnableWriteProtection() function allows the user to enable the write protection for the page(s) of SRAM2. Disabling SRAM2 page(s) protection is performed only by a global peripheral reset or a by a system reset.

This section contains the following APIs:

- *HAL_RAMCFG_EnableWriteProtection()*

63.2.7 Erase Operation Functions

This section provides functions allowing a hardware erase for the given SRAM.

The HAL_RAMCFG_Erase() function allows a hardware mass erase for the given SRAM. The erase value for all SRAMs is 0.

This section contains the following APIs:

- *HAL_RAMCFG_Erase()*

63.2.8 Handle Interrupt and Callbacks Functions

This section provides functions to handle RAMCFG interrupts and Register / UnRegister the different callbacks.

The HAL_RAMCFG_IRQHandler() function allows the user to handle the active RAMCFG interrupt request. The HAL_RAMCFG_RegisterCallback() function allows the user to register the selected RAMCFG callbacks. The HAL_RAMCFG_UnRegisterCallback() function allows the user to unregister the selected RAMCFG callbacks.

This section contains the following APIs:

- *HAL_RAMCFG_IRQHandler()*
- *HAL_RAMCFG_DetectSingleErrorCallback()*
- *HAL_RAMCFG_DetectDoubleErrorCallback()*
- *HAL_RAMCFG_RegisterCallback()*
- *HAL_RAMCFG_UnRegisterCallback()*

63.2.9 State and Error Functions

This section provides functions to check and get the RAMCFG state and the error code.

The HAL_RAMCFG_GetState() function allows the user to get the RAMCFG peripheral state. The HAL_RAMCFG_GetError() function allows the user to get the RAMCFG peripheral error code.

This section contains the following APIs:

- *HAL_RAMCFG_GetState()*
- *HAL_RAMCFG_GetError()*

63.2.10 Detailed description of functions

HAL_RAMCFG_Init

Function name

HAL_StatusTypeDef HAL_RAMCFG_Init (RAMCFG_HandleTypeDef * hramcfg)

Function description

Initialize the RAMCFG by clearing flags and disabling interrupts.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_DeInit

Function name

HAL_StatusTypeDef HAL_RAMCFG_DeInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

DeInitialize the RAMCFG peripheral.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_MspInit

Function name

void HAL_RAMCFG_MspInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

Initialize the RAMCFG MSP.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_MspDeInit

Function name

void HAL_RAMCFG_MspDeInit (RAMCFG_HandleTypeDef * hramcfg)

Function description

DeInitialize the RAMCFG MSP.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_StartECC

Function name

HAL_StatusTypeDef HAL_RAMCFG_StartECC (RAMCFG_HandleTypeDef * hramcfg)

Function description

Start ECC mechanism for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_StopECC

Function name

HAL_StatusTypeDef HAL_RAMCFG_StopECC (RAMCFG_HandleTypeDef * hramcfg)

Function description

Stop ECC mechanism for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_EnableNotification

Function name

HAL_StatusTypeDef HAL_RAMCFG_EnableNotification (RAMCFG_HandleTypeDef * hramcfg, uint32_t Notifications)

Function description

Enable the RAMCFG error interrupts.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **Notifications:** : Select the notification to be enabled. This parameter can be any value of RAMCFG Interrupts group.

Return values

- **HAL:** status.

HAL_RAMCFG_DisableNotification

Function name

HAL_StatusTypeDef HAL_RAMCFG_DisableNotification (RAMCFG_HandleTypeDef * hramcfg, uint32_t Notifications)

Function description

Disable the RAMCFG error interrupts.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **Notifications:** : Select the notification to be disabled. This parameter can be : RAMCFG_IT_SINGLEERR : Single Error Interrupt. RAMCFG_IT_DOUBLEERR : Double Error Interrupt.

Return values

- **HAL:** status.

HAL_RAMCFG_IsECCSingleErrorDetected

Function name

uint32_t HAL_RAMCFG_IsECCSingleErrorDetected (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Check if an ECC single error has occurred.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **State:** of bit (1 or 0).

HAL_RAMCFG_IsECCDoubleErrorDetected

Function name

uint32_t HAL_RAMCFG_IsECCDoubleErrorDetected (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Check if an ECC double error was occurred.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **State:** of bit (1 or 0).

HAL_RAMCFG_GetSingleErrorAddress

Function name

uint32_t HAL_RAMCFG_GetSingleErrorAddress (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG single error address.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **Single:** error address offset.

HAL_RAMCFG_GetDoubleErrorAddress

Function name

uint32_t HAL_RAMCFG_GetDoubleErrorAddress (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG double error address.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **Double:** error address offset.

HAL_RAMCFG_ConfigWaitState

Function name

HAL_StatusTypeDef HAL_RAMCFG_ConfigWaitState (RAMCFG_HandleTypeDef * hramcfg, uint32_t WaitState)

Function description

Configure the wait state value for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **WaitState:** : Select the value of wait state to be configured.

Return values

- **HAL:** status.

HAL_RAMCFG_GetWaitState

Function name

uint32_t HAL_RAMCFG_GetWaitState (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the current wait state value for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **Wait:** state value.

HAL_RAMCFG_EnableWriteProtection

Function name

HAL_StatusTypeDef HAL_RAMCFG_EnableWriteProtection (RAMCFG_HandleTypeDef * hramcfg, uint32_t StartPage, uint32_t NbPage)

Function description

Enable write protection for the given page(s).

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **StartPage:** : Select the start page number.
- **NbPage:** : Number of page to be protected.

Return values

- **HAL:** status.

HAL_RAMCFG_Erase

Function name

HAL_StatusTypeDef HAL_RAMCFG_Erase (RAMCFG_HandleTypeDef * hramcfg)

Function description

Launch a Mass Erase for the given SRAM.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **HAL:** status.

HAL_RAMCFG_IRQHandler

Function name

void HAL_RAMCFG_IRQHandler (RAMCFG_HandleTypeDef * hramcfg)

Function description

Handles RAMCFG interrupt request.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **None.:**

HAL_RAMCFG_DetectSingleErrorCallback

Function name

void HAL_RAMCFG_DetectSingleErrorCallback (RAMCFG_HandleTypeDef * hramcfg)

Function description

RAMCFG single error detection callback.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_DetectDoubleErrorCallback

Function name

void HAL_RAMCFG_DetectDoubleErrorCallback (RAMCFG_HandleTypeDef * hramcfg)

Function description

RAMCFG double error detection callback.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG.

Return values

- **None.:**

HAL_RAMCFG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMCFG_RegisterCallback (RAMCFG_HandleTypeDef * hramcfg, HAL_RAMCFG_CallbackIDTypeDef CallbackID, void(*)(RAMCFG_HandleTypeDef *_hramcfg) pCallback)

Function description

Register RAMCFG callbacks.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **CallbackID:** : User Callback identifier a HAL_RAMCFG_CallbackIDTypeDef ENUM as parameter.
- **pCallback:** : Pointer to private callback function.

Return values

- **HAL:** status.

HAL_RAMCFG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RAMCFG_UnRegisterCallback (RAMCFG_HandleTypeDef * hramcfg, HAL_RAMCFG_CallbackIDTypeDef CallbackID)

Function description

UnRegister RAMCFG callbacks.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.
- **CallbackID:** : User Callback identifier a HAL_RAMCFG_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status.

HAL_RAMCFG_GetError

Function name

uint32_t HAL_RAMCFG_GetError (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG peripheral error code.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **RAMCFG:** error code.

HAL_RAMCFG_GetState

Function name

HAL_RAMCFG_StateTypeDef HAL_RAMCFG_GetState (const RAMCFG_HandleTypeDef * hramcfg)

Function description

Get the RAMCFG peripheral state.

Parameters

- **hramcfg:** : Pointer to a RAMCFG_HandleTypeDef structure that contains the configuration information for the specified RAMCFG instance.

Return values

- **RAMCFG:** state.

63.3 RAMCFG Firmware driver defines

The following section lists the various define and macros of the module.

63.3.1 RAMCFG

RAMCFG

RAMCFG Error Codes

HAL_RAMCFG_ERROR_NONE

RAMCFG No Error

HAL_RAMCFG_ERROR_TIMEOUT

RAMCFG Timeout Error

HAL_RAMCFG_ERROR_BUSY

RAMCFG Busy Error

HAL_RAMCFG_ERROR_INVALID_CALLBACK

Invalid Callback error

RAMCFG Exported Macros

__HAL_RAMCFG_ENABLE_IT

Description:

- Enable the specified RAMCFG interrupts.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __INTERRUPT__: Specifies the RAMCFG interrupt sources to be enabled. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.
  - RAMCFG_IT_NMIERR : Double Error Interrupt redirection to NMI Mask.
  - RAMCFG_IT_ALL : All Interrupt Mask.

Return value:

- None

__HAL_RAMCFG_DISABLE_IT

Description:

- Disable the specified RAMCFG interrupts.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __INTERRUPT__: Specifies the RAMCFG interrupt sources to be disabled. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.

Return value:

- None

Notes:

- This macros is used only to disable RAMCFG_IT_SINGLEERR and RAMCFG_IT_DOUBLEERR interrupts. RAMCFG_IT_NMIERR interrupt can only be disabled by global peripheral reset or system reset.

__HAL_RAMCFG_GET_IT_SOURCE

Description:

- Check whether the specified RAMCFG interrupt source is enabled or not.

Parameters:

- __HANDLE__: : Specifies the RAMCFG Handle.
- __INTERRUPT__: : Specifies the RAMCFG interrupt source to check. This parameter can be one of the following values:
  - RAMCFG_IT_SINGLEERR : Single Error Interrupt Mask.
  - RAMCFG_IT_DOUBLEERR : Double Error Interrupt Mask.
  - RAMCFG_IT_NMIERR : Double Error Interrupt Redirection to NMI Mask.

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RAMCFG_GET_FLAG

Description:

- Get the RAMCFG pending flags.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __FLAG__: : Specifies the flag to be checked. This parameter can be one of the following values:
  - RAMCFG_FLAG_SINGLEERR : Single Error Detected and Corrected Flag.
  - RAMCFG_FLAG_DOUBLEERR : Double Error Detected Flag.
  - RAMCFG_FLAG_SRAMBUSY : SRAM Busy Flag.

Return value:

- The: state of FLAG (SET or RESET).

__HAL_RAMCFG_CLEAR_FLAG

Description:

- Clear the RAMCFG pending flags.

Parameters:

- __HANDLE__: : Specifies RAMCFG handle.
- __FLAG__: : Specifies the flag to be cleared. This parameter can be any combination of the following values:
  - RAMCFG_FLAG_SINGLEERR : Single Error Detected and Corrected Flag.
  - RAMCFG_FLAG_DOUBLEERR : Double Error Detected Flag.

Return value:

- None.

__HAL_RAMCFG_RESET_HANDLE_STATE

Description:

- Reset the RAMCFG handle state.

Parameters:

- __HANDLE__: : Specifies the RAMCFG Handle.

Return value:

- None.

RAMCFG Monitor Flags

RAMCFG_FLAG_SINGLEERR

RAMCFG Single Error Detected and Corrected Flag

RAMCFG_FLAG_DOUBLEERR

RAMCFG Double Error Detected Flag

RAMCFG_FLAG_SRAMBUSY

RAMCFG SRAM busy Flag

RAMCFG_FLAGS_ALL

RAMCFG All Flags

RAMCFG Interrupts

RAMCFG_IT_SINGLEERR

RAMCFG Single Error Interrupt

RAMCFG_IT_DOUBLEERR

RAMCFG Double Error Interrupt

RAMCFG_IT_NMIERR

RAMCFG Double Error redirected to NMI Interrupt

RAMCFG_IT_ALL

RAMCFG All RAMCFG interrupt

RAMCFG Keys

RAMCFG_ERASE_KEY1

RAMCFG launch Erase Key 1

RAMCFG_ERASE_KEY2

RAMCFG launch Erase Key 2

RAMCFG_ECC_KEY1

RAMCFG launch ECC Key 1

RAMCFG_ECC_KEY2

RAMCFG launch ECC Key 2

RAMCFG Wait State

RAMCFG_WAITSTATE_0

RAMCFG 0 Wait State

RAMCFG_WAITSTATE_1

RAMCFG 1 Wait State

RAMCFG_WAITSTATE_2

RAMCFG 2 Wait State

RAMCFG_WAITSTATE_3

RAMCFG 3 Wait State

RAMCFG_WAITSTATE_4

RAMCFG 4 Wait State

RAMCFG_WAITSTATE_5

RAMCFG 5 Wait State

RAMCFG_WAITSTATE_6

RAMCFG 6 Wait State

RAMCFG_WAITSTATE_7

RAMCFG 7 Wait State

64 HAL RCC Generic Driver

64.1 RCC Firmware driver registers structures

64.1.1 RCC_PLLInitTypeDef

RCC_PLLInitTypeDef is defined in the stm32u5xx_hal_rcc.h

Data Fields

- ***uint32_t PLLState***
- ***uint32_t PLLSource***
- ***uint32_t PLLM***
- ***uint32_t PLLMBOOST***
- ***uint32_t PLLN***
- ***uint32_t PLLP***
- ***uint32_t PLLQ***
- ***uint32_t PLLR***
- ***uint32_t PLLRGE***
- ***uint32_t PLLFRACN***

Field Documentation

- ***uint32_t RCC_PLLInitTypeDef::PLLState***
  The new state of the PLL. This parameter can be a value of ***RCC_PLL_Config***
- ***uint32_t RCC_PLLInitTypeDef::PLLSource***
  RCC_PLLSource: PLL entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLLInitTypeDef::PLLM***
  PLLM: Division factor for PLL VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***uint32_t RCC_PLLInitTypeDef::PLLMBOOST***
  PLLMBOOST: Prescaler for EPOD booster input clock. This parameter must be a value of ***RCC_PLLMBOOST_EPOD_Clock_Divider***
- ***uint32_t RCC_PLLInitTypeDef::PLLN***
  PLLN: Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512
- ***uint32_t RCC_PLLInitTypeDef::PLLP***
  PLLP: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLLInitTypeDef::PLLQ***
  PLLQ: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLLInitTypeDef::PLLR***
  PLLR: Division factor for system clock. This parameter must be a number between Min_Data = 2 and Max_Data = 128 Only division by 1 and even division factors are allowed
- ***uint32_t RCC_PLLInitTypeDef::PLLRGE***
  PLLRGE: PLL1 clock Input range This parameter must be a value of ***RCC_PLL_VCI_Range***
- ***uint32_t RCC_PLLInitTypeDef::PLLFRACN***
  PLLFRACN: Specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

64.1.2 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32u5xx_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***

- ***uint32_t HSIState***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***
- ***uint32_t LSIDiv***
- ***uint32_t MSIState***
- ***uint32_t MSICalibrationValue***
- ***uint32_t MSIClockRange***
- ***uint32_t MSIKClockRange***
- ***uint32_t HSI48State***
- ***uint32_t SHSIState***
- ***uint32_t MSIKState***
- ***RCC_PLLInitTypeDef PLL***

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
  The oscillators to be configured. This parameter can be a value of ***RCC_Oscillator_Type***
- ***uint32_t RCC_OscInitTypeDef::HSEState***
  The new state of the HSE. This parameter can be a value of ***RCC_HSE_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEState***
  The new state of the LSE. This parameter can be a value of ***RCC_LSE_Config***
- ***uint32_t RCC_OscInitTypeDef::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_Config***
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
  The calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F on the other devices
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::LSIDiv***
  The division factor of the LSI. This parameter can be a value of ***RCC_LSI_Div***
- ***uint32_t RCC_OscInitTypeDef::MSIState***
  The new state of the MSI. This parameter can be a value of ***RCC_MSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSICalibrationValue***
  The calibration trimming value (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t RCC_OscInitTypeDef::MSIClockRange***
  The MSI frequency range. This parameter can be a value of ***RCC_MSI_Clock_Range***
- ***uint32_t RCC_OscInitTypeDef::MSIKClockRange***
  The MSIK frequency range. This parameter can be a value of ***RCC_MSIK_Clock_Range***
- ***uint32_t RCC_OscInitTypeDef::HSI48State***
  The new state of the HSI48. This parameter can be a value of ***RCC_HSI48_Config***
- ***uint32_t RCC_OscInitTypeDef::SHSIState***
  The new state of the SHSI. This parameter can be a value of ***RCC_SHSI_Config***
- ***uint32_t RCC_OscInitTypeDef::MSIKState***
  The new state of the MSIK. This parameter can be a value of ***RCC_MSIK_Config***
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
  Main PLL structure parameters

64.1.3 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32u5xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***

- ***uint32_t APB2CLKDivider***
- ***uint32_t APB3CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a value of ***RCC_System_Clock_Type***
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
  The clock source used as system clock (SYSCLK). This parameter can be a value of ***RCC_System_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_AHB_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_APB3_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_APB3_Clock_Source***
- ***uint32_t RCC_ClkInitTypeDef::APB3CLKDivider***
  The APB3 clock (PCLK3) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_APB1_APB2_APB3_Clock_Source***

64.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

64.2.1 RCC specific features

After reset the device is running from Multiple Speed Internal oscillator (4 MHz) with Flash 0 wait state. Flash prefetch buffer, D-Cache and I-Cache are disabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHBs) and Low speed (APBs) busses: all peripherals mapped on these busses are running at MSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analog mode, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals which clocks are not derived from the System clock (SAIx, SYSTICK, RTC, ADC, USB OTG FS/USB FS/SDMMC1/RNG)

64.2.2 Initialization and de-initialization functions

This section provides functions allowing to configure the internal and external oscillators (HSE, HSI, LSE, MSI, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

- HSI (high-speed internal): 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
- MSI (Multiple Speed Internal): Its frequency is software trimmable from 100KHZ to 48MHZ. It can be used to generate the clock for the USB FS or USB OTG FS (48 MHz). The number of flash wait states is automatically adjusted when MSI range is updated with HAL_RCC_OscConfig() and the MSI is used as System clock source.
- LSI (low-speed internal): 32 KHz low consumption RC used as IWDG and/or RTC clock source.
- HSE (high-speed external): 4 to 48 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also optionally as RTC clock source.

- LSE (low-speed external): 32.768 KHz oscillator used optionally as RTC clock source.
- PLL (clocked by HSI, HSE or MSI) providing up to three independent output clocks:
  – The first output is used to generate the high speed system clock (up to 80MHz).
  – The second output is used to generate the clock for the USB FS or USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDMMC1 (<= 48 MHz).
  – The third output is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
- PLL2 (clocked by HSI, HSE or MSI) providing up to three independent output clocks:
  – The first output is used to generate SAR ADC1 clock.
  – The second output is used to generate the clock for the USB Fs or USB OTG FS (48 MHz), the random analog generator (<=48 MHz) and the SDMMC1 (<= 48 MHz).
  – The Third output is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
- PLL3 (clocked by HSI , HSE or MSI) providing up to two independent output clocks:
  – The first output is used to generate SAR ADC4 clock.
  – The second output is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
- CSS (Clock security system): once enabled, if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to HSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
- MCO (microcontroller clock output): used to output MSI, LSI, HSI, LSE, HSE or main PLL clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

- Several clock sources can be used to drive the System clock (SYSCLK): MSI, HSI, HSE and main PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks.

Note: All the peripheral clocks are derived from the System clock (SYSCLK) except:

- *SAI: the SAI clock can be derived either from a specific PLL (PLL2) or (PLL3) or from an external clock mapped on the SAI_CKIN pin. You have to use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use __HAL_RCC_RTC_ENABLE() and HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *USB FS, USB OTG FS, SDMMC1 and RNG: USB OTG FS or USB FS requires a frequency equal to 48 MHz to work correctly, while the SDMMC1 and RNG peripherals require a frequency equal or lower than to 48 MHz. This clock is derived of the main PLL or PLL2 through PLLQ divider. You have to enable the peripheral clock and use HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *IWDG clock which is always the LSI clock.*

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2 is 80 MHz. The clock source frequency should be adapted depending on the device voltage range as listed in the Reference Manual "Clock source frequency versus voltage scaling" chapter.

This section contains the following APIs:

- ***HAL_RCC_DeInit()***
- ***HAL_RCC_OscConfig()***
- ***HAL_RCC_ClockConfig()***

64.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to:

- Output clock to MCO pin.

- Retrieve current clock frequencies.
- Enable the Clock Security System.

This section contains the following APIs:
- *HAL_RCC_MCOConfig()*
- *HAL_RCC_GetSysClockFreq()*
- *HAL_RCC_GetHCLKFreq()*
- *HAL_RCC_GetPCLK1Freq()*
- *HAL_RCC_GetPCLK2Freq()*
- *HAL_RCC_GetPCLK3Freq()*
- *HAL_RCC_GetOscConfig()*
- *HAL_RCC_GetClockConfig()*
- *HAL_RCC_GetResetSource()*
- *HAL_RCC_EnableCSS()*
- *HAL_RCC_NMI_IRQHandler()*
- *HAL_RCC_CSSCallback()*

64.2.4 RCC attributes functions

This section contains the following APIs:
- *HAL_RCC_ConfigAttributes()*

64.2.5 Detailed description of functions

HAL_RCC_DeInit

Function name

HAL_StatusTypeDef HAL_RCC_DeInit (void)

Function description

Reset the RCC clock configuration to the default reset state.

Return values

- **HAL:** status

Notes

- The default reset state of the clock configuration is given below: MSI ON and used as system clock sourceHSE, HSI, PLL, PLL2 and PLLISAI2 OFFAHB, APB1 and APB2 prescaler set to 1CSS, MCO1 OFFAll interrupts disabledAll interrupt and reset flags cleared
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (const RCC_OscInitTypeDef * pRCC_OscInitStruct)

Function description

Initialize the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **pRCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- The PLL is not disabled when used as system clock.
- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this function. User should request a transition to LSE Off first and then LSE On or LSE Bypass.
- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this function. User should request a transition to HSE Off first and then HSE On or HSE Bypass.

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (const RCC_ClkInitTypeDef *const pRCC_ClkInitStruct, uint32_t FLatency)

Function description

Initialize the CPU, AHB and APB busses clocks according to the specified parameters in the pRCC_ClkInitStruct.

Parameters

- **pRCC_ClkInitStruct:** pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC peripheral.
- **FLatency:** FLASH Latency This parameter can be one of the following values:
  - FLASH_LATENCY_0 FLASH 0 Latency cycle
  - FLASH_LATENCY_1 FLASH 1 Latency cycle
  - FLASH_LATENCY_2 FLASH 2 Latency cycles
  - FLASH_LATENCY_3 FLASH 3 Latency cycles
  - FLASH_LATENCY_4 FLASH 4 Latency cycles
  - FLASH_LATENCY_5 FLASH 5 Latency cycles

Return values

- **HAL:** status

Notes

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function
- The MSI is used by default as system clock source after startup from Reset, wake-up from STANDBY mode. After restart from Reset, the MSI frequency is set to its default value 4 MHz.
- The HSI can be selected as system clock source after wakeup from STOP modes or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source is ready.
- You can use HAL_RCC_GetClockConfig() function to know which clock is currently used as system clock source.
- Depending on the device voltage range, the software has to set correctly HPRE[3:0] bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

HAL_RCC_MCOConfig

Function name

void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)

Function description

Select the clock source to output on MCO pin(PA8).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source. For STM32U5xx family this parameter can have only one value:
  – RCC_MCO1 Clock source to output on MCO1 pin(PA8).
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  – RCC_MCO1SOURCE_NOCLOCK MCO output disabled, no clock on MCO
  – RCC_MCO1SOURCE_SYSCLK system clock selected as MCO source
  – RCC_MCO1SOURCE_MSI MSI clock selected as MCO source
  – RCC_MCO1SOURCE_HSI HSI clock selected as MCO source
  – RCC_MCO1SOURCE_HSE HSE clock selected as MCO source
  – RCC_MCO1SOURCE_PLL1CLK main PLL clock selected as MCO source
  – RCC_MCO1SOURCE_LSI LSI clock selected as MCO source
  – RCC_MCO1SOURCE_LSE LSE clock selected as MCO source
  – RCC_MCO1SOURCE_HSI48 HSI48 clock selected as MCO source for devices with HSI48
- **RCC_MCODiv:** specifies the MCO prescaler. This parameter can be one of the following values:
  – RCC_MCODIV_1 no division applied to MCO clock
  – RCC_MCODIV_2 division by 2 applied to MCO clock
  – RCC_MCODIV_4 division by 4 applied to MCO clock
  – RCC_MCODIV_8 division by 8 applied to MCO clock
  – RCC_MCODIV_16 division by 16 applied to MCO clock

Return values

- **None:**

Notes

- PA8 should be configured in alternate function mode.

HAL_RCC_EnableCSS

Function name

void HAL_RCC_EnableCSS (void)

Function description

Enable the Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M4 NMI (Non-Maskable Interrupt) exception vector.
- The Clock Security System can only be cleared by reset.

HAL_RCC_GetSysClockFreq

Function name

uint32_t HAL_RCC_GetSysClockFreq (void)

Function description

Return the SYSCLK frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is MSI, function returns values based on MSI Value as defined by the MSI range.
- If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)
- If SYSCLK source is PLL, function returns values based on HSE_VALUE(**), HSI_VALUE(*) or MSI Value multiplied/divided by the PLL factors.
- (*) HSI_VALUE is a constant defined in stm32u5xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) HSE_VALUE is a constant defined in stm32u5xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
- The result of this function could be not correct when using fractional value for HSE crystal.
- This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters.
- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetHCLKFreq

Function name

uint32_t HAL_RCC_GetHCLKFreq (void)

Function description

Return the HCLK frequency.

Return values

- **HCLK:** frequency in Hz

Notes

- Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect.
- The SystemCoreClock CMSIS variable is used to store System Clock Frequency.

HAL_RCC_GetPCLK1Freq

Function name

uint32_t HAL_RCC_GetPCLK1Freq (void)

Function description

Return the PCLK1 frequency.

Return values

- **PCLK1:** frequency in Hz

Notes

- Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK2Freq

Function name

uint32_t HAL_RCC_GetPCLK2Freq (void)

Function description

Return the PCLK2 frequency.

Return values

- **PCLK2:** frequency in Hz

Notes

- Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetPCLK3Freq

Function name

uint32_t HAL_RCC_GetPCLK3Freq (void)

Function description

Return the PCLK3 frequency.

Return values

- **PCLK3:** frequency in Hz

Notes

- Each time PCLK3 changes, this function must be called to update the right PCLK3 value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCC_GetOscConfig

Function name

void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * pRCC_OscInitStruct)

Function description

Get the pRCC_OscInitStruct according to the internal RCC configuration registers.

Parameters

- **pRCC_OscInitStruct:** pointer to an RCC_OscInitTypeDef structure that will be configured.

Return values

- **None:**

HAL_RCC_GetClockConfig

Function name

void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * pRCC_ClkInitStruct, uint32_t * pFLatency)

Function description

Configure the pRCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **pRCC_ClkInitStruct:** pointer to an RCC_ClkInitTypeDef structure that will be configured.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

HAL_RCC_GetResetSource

Function name

uint32_t HAL_RCC_GetResetSource (void)

Function description

Get and clear reset flags.

Return values

- **can:** be a combination of Reset Flag

Notes

- Once reset flags are retrieved, this API is clearing them in order to isolate next reset reason.

HAL_RCC_NMI_IRQHandler

Function name

void HAL_RCC_NMI_IRQHandler (void)

Function description

Handle the RCC Clock Security System interrupt request.

Return values

- **None:**

Notes

- This API should be called under the NMI_Handler().

HAL_RCC_CSSCallback

Function name

void HAL_RCC_CSSCallback (void)

Function description

RCC Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCC_ConfigAttributes

Function name

void HAL_RCC_ConfigAttributes (uint32_t Item, uint32_t Attributes)

Function description

Configure the RCC item attribute(s).

Parameters

- **Item:** Item(s) to set attributes on. This parameter can be a one or a combination of RCC items
- **Attributes:** specifies the RCC secure/privilege attributes. This parameter can be a value of RCC attributes

Return values

- **None:**

Notes

- Available attributes are to secure items and set RCC as privileged.

HAL_RCC_GetConfigAttributes

Function name

HAL_StatusTypeDef HAL_RCC_GetConfigAttributes (uint32_t Item, uint32_t * pAttributes)

Function description

Get the attribute of a RCC item.

Parameters

- **Item:** Single item to get secure/non-secure and privilege/non-privilege attribute from. This parameter can be a one value of RCC items except RCC_ALL.
- **pAttributes:** pointer to return the attributes.

Return values

- **HAL:** Status.

Notes

- Secure and non-secure attributes are only available from secure state when the system implements the security (TZEN=1)

64.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

64.3.1 RCC

RCC

AHB1 Peripheral Force Release Reset

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_GPDMA1_FORCE_RESET

__HAL_RCC_CORDIC_FORCE_RESET

__HAL_RCC_FMAC_FORCE_RESET

__HAL_RCC_MDF1_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_JPEG_FORCE_RESET

__HAL_RCC_TSC_FORCE_RESET

__HAL_RCC_RAMCFG_FORCE_RESET

__HAL_RCC_DMA2D_FORCE_RESET

__HAL_RCC_GFXMMU_FORCE_RESET

__HAL_RCC_GPU2D_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_GPDMA1_RELEASE_RESET

__HAL_RCC_CORDIC_RELEASE_RESET

__HAL_RCC_FMAC_RELEASE_RESET

__HAL_RCC_MDF1_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_JPEG_RELEASE_RESET

__HAL_RCC_TSC_RELEASE_RESET

__HAL_RCC_RAMCFG_RELEASE_RESET

__HAL_RCC_DMA2D_RELEASE_RESET

__HAL_RCC_GFXMMU_RELEASE_RESET

__HAL_RCC_GPU2D_RELEASE_RESET

AHB1 Peripheral Clock Enable Disable

__HAL_RCC_GPDMA1_CLK_ENABLE

__HAL_RCC_CORDIC_CLK_ENABLE

__HAL_RCC_FMAC_CLK_ENABLE

__HAL_RCC_TSC_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_JPEG_CLK_ENABLE

__HAL_RCC_RAMCFG_CLK_ENABLE

__HAL_RCC_FLASH_CLK_ENABLE

__HAL_RCC_MDF1_CLK_ENABLE

__HAL_RCC_DMA2D_CLK_ENABLE

__HAL_RCC_GFXMMU_CLK_ENABLE

__HAL_RCC_GPU2D_CLK_ENABLE

__HAL_RCC_DCACHE2_CLK_ENABLE

__HAL_RCC_GTZC1_CLK_ENABLE

__HAL_RCC_BKPSRAM_CLK_ENABLE

__HAL_RCC_DCACHE1_CLK_ENABLE

__HAL_RCC_SRAM1_CLK_ENABLE

__HAL_RCC_GPDMA1_CLK_DISABLE

__HAL_RCC_CORDIC_CLK_DISABLE

__HAL_RCC_FMAC_CLK_DISABLE

__HAL_RCC_MDF1_CLK_DISABLE

__HAL_RCC_FLASH_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

__HAL_RCC_JPEG_CLK_DISABLE

__HAL_RCC_TSC_CLK_DISABLE

__HAL_RCC_RAMCFG_CLK_DISABLE

__HAL_RCC_DMA2D_CLK_DISABLE

__HAL_RCC_GFXMMU_CLK_DISABLE

__HAL_RCC_GPU2D_CLK_DISABLE

__HAL_RCC_DCACHE2_CLK_DISABLE

__HAL_RCC_GTZC1_CLK_DISABLE

__HAL_RCC_BKPSRAM_CLK_DISABLE

__HAL_RCC_DCACHE1_CLK_DISABLE

__HAL_RCC_SRAM1_CLK_DISABLE

AHB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPDMA1_IS_CLK_ENABLED

__HAL_RCC_CORDIC_IS_CLK_ENABLED

__HAL_RCC_FMAC_IS_CLK_ENABLED

__HAL_RCC_MDF1_IS_CLK_ENABLED

__HAL_RCC_FLASH_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_JPEG_IS_CLK_ENABLED

__HAL_RCC_TSC_IS_CLK_ENABLED

__HAL_RCC_RAMCFG_IS_CLK_ENABLED

__HAL_RCC_DMA2D_IS_CLK_ENABLED

__HAL_RCC_GFXMMU_IS_CLK_ENABLED

__HAL_RCC_GPU2D_IS_CLK_ENABLED

__HAL_RCC_DCACHE2_IS_CLK_ENABLED

__HAL_RCC_GTZC1_IS_CLK_ENABLED

__HAL_RCC_BKPSRAM_IS_CLK_ENABLED

__HAL_RCC_DCACHE1_IS_CLK_ENABLED

__HAL_RCC_SRAM1_IS_CLK_ENABLED

__HAL_RCC_GPDMA1_IS_CLK_DISABLED

__HAL_RCC_CORDIC_IS_CLK_DISABLED

__HAL_RCC_FMAC_IS_CLK_DISABLED

__HAL_RCC_MDF1_IS_CLK_DISABLED

__HAL_RCC_FLASH_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

__HAL_RCC_JPEG_IS_CLK_DISABLED

__HAL_RCC_TSC_IS_CLK_DISABLED

__HAL_RCC_RAMCFG_IS_CLK_DISABLED

__HAL_RCC_DMA2D_IS_CLK_DISABLED

__HAL_RCC_GFXMMU_IS_CLK_DISABLED

__HAL_RCC_GPU2D_IS_CLK_DISABLED

__HAL_RCC_DCACHE2_IS_CLK_DISABLED

__HAL_RCC_GTZC1_IS_CLK_DISABLED

__HAL_RCC_BKPSRAM_IS_CLK_DISABLED

__HAL_RCC_DCACHE1_IS_CLK_DISABLED

__HAL_RCC_SRAM1_IS_CLK_DISABLED

AHB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPDMA1_CLK_SLEEP_ENABLE

__HAL_RCC_CORDIC_CLK_SLEEP_ENABLE

__HAL_RCC_FMAC_CLK_SLEEP_ENABLE

__HAL_RCC_MDF1_CLK_SLEEP_ENABLE

__HAL_RCC_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_JPEG_CLK_SLEEP_ENABLE

__HAL_RCC_TSC_CLK_SLEEP_ENABLE

__HAL_RCC_RAMCFG_CLK_SLEEP_ENABLE

__HAL_RCC_DMA2D_CLK_SLEEP_ENABLE

__HAL_RCC_GFXMMU_CLK_SLEEP_ENABLE

__HAL_RCC_GPU2D_CLK_SLEEP_ENABLE

__HAL_RCC_DCACHE2_CLK_SLEEP_ENABLE

__HAL_RCC_GTZC1_CLK_SLEEP_ENABLE

__HAL_RCC_BKPSRAM_CLK_SLEEP_ENABLE

__HAL_RCC_ICACHE_CLK_SLEEP_ENABLE

__HAL_RCC_DCACHE1_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM1_CLK_SLEEP_ENABLE

__HAL_RCC_GPDMA1_CLK_SLEEP_DISABLE

__HAL_RCC_CORDIC_CLK_SLEEP_DISABLE

__HAL_RCC_FMAC_CLK_SLEEP_DISABLE

__HAL_RCC_MDF1_CLK_SLEEP_DISABLE

__HAL_RCC_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

__HAL_RCC_JPEG_CLK_SLEEP_DISABLE

__HAL_RCC_TSC_CLK_SLEEP_DISABLE

__HAL_RCC_RAMCFG_CLK_SLEEP_DISABLE

__HAL_RCC_DMA2D_CLK_SLEEP_DISABLE

__HAL_RCC_GFXMMU_CLK_SLEEP_DISABLE

__HAL_RCC_GPU2D_CLK_SLEEP_DISABLE

__HAL_RCC_DCACHE2_CLK_SLEEP_DISABLE

__HAL_RCC_GTZC1_CLK_SLEEP_DISABLE

__HAL_RCC_BKPSRAM_CLK_SLEEP_DISABLE

__HAL_RCC_ICACHE_CLK_SLEEP_DISABLE

__HAL_RCC_DCACHE1_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM1_CLK_SLEEP_DISABLE

AHB2 Peripheral Force Release Reset

__HAL_RCC_AHB2_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOE_FORCE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_GPIOG_FORCE_RESET

__HAL_RCC_GPIOH_FORCE_RESET

__HAL_RCC_GPIOI_FORCE_RESET

__HAL_RCC_GPIOJ_FORCE_RESET

__HAL_RCC_ADC12_FORCE_RESET

__HAL_RCC_DCMI_PSSI_FORCE_RESET

__HAL_RCC_USB_OTG_HS_FORCE_RESET

__HAL_RCC_AES_FORCE_RESET

__HAL_RCC_HASH_FORCE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_PKA_FORCE_RESET

__HAL_RCC_SAES_FORCE_RESET

__HAL_RCC_OSPIM_FORCE_RESET

__HAL_RCC_OTFDEC1_FORCE_RESET

__HAL_RCC_OTFDEC2_FORCE_RESET

__HAL_RCC_SDMMC1_FORCE_RESET

__HAL_RCC_SDMMC2_FORCE_RESET

__HAL_RCC_FMC_FORCE_RESET

__HAL_RCC_OSPI1_FORCE_RESET

__HAL_RCC_OSPI2_FORCE_RESET

__HAL_RCC_HSPI1_FORCE_RESET

__HAL_RCC_AHB2_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOE_RELEASE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

__HAL_RCC_GPIOG_RELEASE_RESET

__HAL_RCC_GPIOH_RELEASE_RESET

__HAL_RCC_GPIOI_RELEASE_RESET

__HAL_RCC_GPIOJ_RELEASE_RESET

__HAL_RCC_ADC12_RELEASE_RESET

__HAL_RCC_DCMI_PSSI_RELEASE_RESET

__HAL_RCC_USB_OTG_HS_RELEASE_RESET

__HAL_RCC_AES_RELEASE_RESET

__HAL_RCC_HASH_RELEASE_RESET

__HAL_RCC_RNG_RELEASE_RESET

__HAL_RCC_PKA_RELEASE_RESET

__HAL_RCC_SAES_RELEASE_RESET

__HAL_RCC_OSPIM_RELEASE_RESET

__HAL_RCC_OTFDEC1_RELEASE_RESET

__HAL_RCC_OTFDEC2_RELEASE_RESET

__HAL_RCC_SDMMC1_RELEASE_RESET

__HAL_RCC_SDMMC2_RELEASE_RESET

__HAL_RCC_FMC_RELEASE_RESET

__HAL_RCC_OSPI1_RELEASE_RESET

__HAL_RCC_OSPI2_RELEASE_RESET

__HAL_RCC_HSPI1_RELEASE_RESET

AHB2 Peripheral Clock Enable Disable

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_GPIOC_CLK_ENABLE

__HAL_RCC_GPIOD_CLK_ENABLE

__HAL_RCC_GPIOE_CLK_ENABLE

__HAL_RCC_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOG_CLK_ENABLE

__HAL_RCC_GPIOH_CLK_ENABLE

__HAL_RCC_GPIOI_CLK_ENABLE

__HAL_RCC_GPIOJ_CLK_ENABLE

__HAL_RCC_ADC12_CLK_ENABLE

__HAL_RCC_DCMI_PSSI_CLK_ENABLE

__HAL_RCC_USB_OTG_HS_CLK_ENABLE

__HAL_RCC_USBPHYC_CLK_ENABLE

__HAL_RCC_AES_CLK_ENABLE

__HAL_RCC_HASH_CLK_ENABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_PKA_CLK_ENABLE

__HAL_RCC_SAES_CLK_ENABLE

__HAL_RCC_OSPIM_CLK_ENABLE

__HAL_RCC_OTFDEC1_CLK_ENABLE

__HAL_RCC_OTFDEC2_CLK_ENABLE

__HAL_RCC_SDMMC1_CLK_ENABLE

__HAL_RCC_SDMMC2_CLK_ENABLE

__HAL_RCC_SRAM2_CLK_ENABLE

__HAL_RCC_SRAM3_CLK_ENABLE

__HAL_RCC_FMC_CLK_ENABLE

__HAL_RCC_OSPI1_CLK_ENABLE

__HAL_RCC_OSPI2_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOE_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_DISABLE

__HAL_RCC_GPIOG_CLK_DISABLE

__HAL_RCC_GPIOH_CLK_DISABLE

__HAL_RCC_GPIOI_CLK_DISABLE

__HAL_RCC_GPIOJ_CLK_DISABLE

__HAL_RCC_ADC12_CLK_DISABLE

__HAL_RCC_DCMI_PSSI_CLK_DISABLE

__HAL_RCC_USB_OTG_HS_CLK_DISABLE

__HAL_RCC_USBPHYC_CLK_DISABLE

__HAL_RCC_AES_CLK_DISABLE

__HAL_RCC_HASH_CLK_DISABLE

__HAL_RCC_RNG_CLK_DISABLE

__HAL_RCC_PKA_CLK_DISABLE

__HAL_RCC_SAES_CLK_DISABLE

__HAL_RCC_OSPIM_CLK_DISABLE

__HAL_RCC_OTFDEC1_CLK_DISABLE

__HAL_RCC_OTFDEC2_CLK_DISABLE

__HAL_RCC_SDMMC1_CLK_DISABLE

__HAL_RCC_SDMMC2_CLK_DISABLE

__HAL_RCC_SRAM2_CLK_DISABLE

__HAL_RCC_SRAM3_CLK_DISABLE

__HAL_RCC_HSPI1_CLK_ENABLE

__HAL_RCC_SRAM6_CLK_ENABLE

__HAL_RCC_SRAM5_CLK_ENABLE

__HAL_RCC_FMC_CLK_DISABLE

__HAL_RCC_OSPI1_CLK_DISABLE

__HAL_RCC_OSPI2_CLK_DISABLE

__HAL_RCC_HSPI1_CLK_DISABLE

__HAL_RCC_SRAM6_CLK_DISABLE

__HAL_RCC_SRAM5_CLK_DISABLE

AHB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_GPIOC_IS_CLK_ENABLED

__HAL_RCC_GPIOD_IS_CLK_ENABLED

__HAL_RCC_GPIOE_IS_CLK_ENABLED

__HAL_RCC_GPIOF_IS_CLK_ENABLED

__HAL_RCC_GPIOG_IS_CLK_ENABLED

__HAL_RCC_GPIOH_IS_CLK_ENABLED

__HAL_RCC_GPIOI_IS_CLK_ENABLED

__HAL_RCC_GPIOJ_IS_CLK_ENABLED

__HAL_RCC_ADC12_IS_CLK_ENABLED

__HAL_RCC_DCMI_PSSI_IS_CLK_ENABLED

__HAL_RCC_USB_OTG_HS_IS_CLK_ENABLED

__HAL_RCC_AES_IS_CLK_ENABLED

__HAL_RCC_HASH_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_PKA_IS_CLK_ENABLED

__HAL_RCC_SAES_IS_CLK_ENABLED

__HAL_RCC_OSPIM_IS_CLK_ENABLED

__HAL_RCC_OTFDEC1_IS_CLK_ENABLED

__HAL_RCC_OTFDEC2_IS_CLK_ENABLED

__HAL_RCC_SDMMC1_IS_CLK_ENABLED

__HAL_RCC_SDMMC2_IS_CLK_ENABLED

__HAL_RCC_SRAM2_IS_CLK_ENABLED

__HAL_RCC_SRAM3_IS_CLK_ENABLED

__HAL_RCC_FMC_IS_CLK_ENABLED

__HAL_RCC_OSPI1_IS_CLK_ENABLED

__HAL_RCC_OSPI2_IS_CLK_ENABLED

__HAL_RCC_HSPI1_IS_CLK_ENABLED

__HAL_RCC_SRAM6_IS_CLK_ENABLED

__HAL_RCC_SRAM5_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_GPIOC_IS_CLK_DISABLED

__HAL_RCC_GPIOD_IS_CLK_DISABLED

__HAL_RCC_GPIOE_IS_CLK_DISABLED

__HAL_RCC_GPIOF_IS_CLK_DISABLED

__HAL_RCC_GPIOG_IS_CLK_DISABLED

__HAL_RCC_GPIOH_IS_CLK_DISABLED

__HAL_RCC_GPIOI_IS_CLK_DISABLED

__HAL_RCC_GPIOJ_IS_CLK_DISABLED

__HAL_RCC_ADC12_IS_CLK_DISABLED

__HAL_RCC_DCMI_PSSI_IS_CLK_DISABLED

__HAL_RCC_USB_OTG_HS_IS_CLK_DISABLED

__HAL_RCC_AES_IS_CLK_DISABLED

__HAL_RCC_HASH_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

__HAL_RCC_PKA_IS_CLK_DISABLED

__HAL_RCC_SAES_IS_CLK_DISABLED

__HAL_RCC_OSPIM_IS_CLK_DISABLED

__HAL_RCC_OTFDEC1_IS_CLK_DISABLED

__HAL_RCC_OTFDEC2_IS_CLK_DISABLED

__HAL_RCC_SDMMC1_IS_CLK_DISABLED

__HAL_RCC_SDMMC2_IS_CLK_DISABLED

__HAL_RCC_SRAM2_IS_CLK_DISABLED

__HAL_RCC_SRAM3_IS_CLK_DISABLED

__HAL_RCC_FMC_IS_CLK_DISABLED

__HAL_RCC_OSPI1_IS_CLK_DISABLED

__HAL_RCC_OSPI2_IS_CLK_DISABLED

__HAL_RCC_HSPI1_IS_CLK_DISABLED

__HAL_RCC_SRAM6_IS_CLK_DISABLED

__HAL_RCC_SRAM5_IS_CLK_DISABLED

AHB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOI_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOJ_CLK_SLEEP_ENABLE

__HAL_RCC_ADC12_CLK_SLEEP_ENABLE

__HAL_RCC_DCMI_PSSI_CLK_SLEEP_ENABLE

__HAL_RCC_USB_OTG_HS_CLK_SLEEP_ENABLE

__HAL_RCC_USBPHYCCLK_SLEEP_ENABLE

__HAL_RCC_AES_CLK_SLEEP_ENABLE

__HAL_RCC_HASH_CLK_SLEEP_ENABLE

__HAL_RCC_RNG_CLK_SLEEP_ENABLE

__HAL_RCC_PKA_CLK_SLEEP_ENABLE

__HAL_RCC_SAES_CLK_SLEEP_ENABLE

__HAL_RCC_OCTOSPIM_CLK_SLEEP_ENABLE

__HAL_RCC_OTFDEC1_CLK_SLEEP_ENABLE

__HAL_RCC_OTFDEC2_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_ENABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM2_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM3_CLK_SLEEP_ENABLE

__HAL_RCC_FMC_CLK_SLEEP_ENABLE

__HAL_RCC_OSPI1_CLK_SLEEP_ENABLE

__HAL_RCC_OSPI2_CLK_SLEEP_ENABLE

__HAL_RCC_HSPI1_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM6_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM5_CLK_SLEEP_ENABLE

__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOI_CLK_SLEEP_DISABLE

__HAL_RCC_GPIOJ_CLK_SLEEP_DISABLE

__HAL_RCC_ADC12_CLK_SLEEP_DISABLE

__HAL_RCC_DCMI_PSSI_CLK_SLEEP_DISABLE

__HAL_RCC_USB_OTG_HS_CLK_SLEEP_DISABLE

__HAL_RCC_USBPHYCCLK_SLEEP_DISABLE

__HAL_RCC_AES_CLK_SLEEP_DISABLE

__HAL_RCC_HASH_CLK_SLEEP_DISABLE

__HAL_RCC_RNG_CLK_SLEEP_DISABLE

__HAL_RCC_PKA_CLK_SLEEP_DISABLE

__HAL_RCC_SAES_CLK_SLEEP_DISABLE

__HAL_RCC_OCTOSPIM_CLK_SLEEP_DISABLE

__HAL_RCC_OTFDEC1_CLK_SLEEP_DISABLE

__HAL_RCC_OTFDEC2_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC1_CLK_SLEEP_DISABLE

__HAL_RCC_SDMMC2_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM2_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM3_CLK_SLEEP_DISABLE

__HAL_RCC_FMC_CLK_SLEEP_DISABLE

__HAL_RCC_OSPI1_CLK_SLEEP_DISABLE

__HAL_RCC_OSPI2_CLK_SLEEP_DISABLE

__HAL_RCC_HSPI1_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM6_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM5_CLK_SLEEP_DISABLE

AHB3ENR Peripheral Clock Enable Disable

__HAL_RCC_LPGPIO1_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_ADC4_CLK_ENABLE

__HAL_RCC_DAC1_CLK_ENABLE

__HAL_RCC_LPDMA1_CLK_ENABLE

__HAL_RCC_ADF1_CLK_ENABLE

__HAL_RCC_GTZC2_CLK_ENABLE

__HAL_RCC_SRAM4_CLK_ENABLE

__HAL_RCC_LPGPIO1_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

__HAL_RCC_ADC4_CLK_DISABLE

__HAL_RCC_DAC1_CLK_DISABLE

__HAL_RCC_LPDMA1_CLK_DISABLE

__HAL_RCC_ADF1_CLK_DISABLE

__HAL_RCC_GTZC2_CLK_DISABLE

__HAL_RCC_SRAM4_CLK_DISABLE

AHB3SMENR Peripheral Clock Sleep Enable Disable

__HAL_RCC_LPGPIO1_CLK_SLEEP_ENABLE

__HAL_RCC_PWR_CLK_SLEEP_ENABLE

__HAL_RCC_ADC4_CLK_SLEEP_ENABLE

__HAL_RCC_DAC1_CLK_SLEEP_ENABLE

__HAL_RCC_LPDMA1_CLK_SLEEP_ENABLE

__HAL_RCC_ADF1_CLK_SLEEP_ENABLE

__HAL_RCC_GTZC2_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM4_CLK_SLEEP_ENABLE

__HAL_RCC_LPGPIO1_CLK_SLEEP_DISABLE

__HAL_RCC_PWR_CLK_SLEEP_DISABLE

__HAL_RCC_ADC4_CLK_SLEEP_DISABLE

__HAL_RCC_DAC1_CLK_SLEEP_DISABLE

__HAL_RCC_LPDMA1_CLK_SLEEP_DISABLE

__HAL_RCC_ADF1_CLK_SLEEP_DISABLE

__HAL_RCC_GTZC2_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM4_CLK_SLEEP_DISABLE

AHB3 Peripheral Force Release Reset

__HAL_RCC_AHB3_FORCE_RESET

__HAL_RCC_LPGPIO1_FORCE_RESET

__HAL_RCC_ADC4_FORCE_RESET

__HAL_RCC_DAC1_FORCE_RESET

__HAL_RCC_LPDMA1_FORCE_RESET

__HAL_RCC_ADF1_FORCE_RESET

__HAL_RCC_AHB3_RELEASE_RESET

__HAL_RCC_LPGPIO1_RELEASE_RESET

__HAL_RCC_ADC4_RELEASE_RESET

__HAL_RCC_DAC1_RELEASE_RESET

__HAL_RCC_LPDMA1_RELEASE_RESET

__HAL_RCC_ADF1_RELEASE_RESET

AHB3 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_LPGPIO1_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_ADC4_IS_CLK_ENABLED

__HAL_RCC_DAC1_IS_CLK_ENABLED

__HAL_RCC_LPDMA1_IS_CLK_ENABLED

__HAL_RCC_ADF1_IS_CLK_ENABLED

__HAL_RCC_GTZC2_IS_CLK_ENABLED

__HAL_RCC_SRAM4_IS_CLK_ENABLED

__HAL_RCC_LPGPIO1_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

__HAL_RCC_ADC4_IS_CLK_DISABLED

__HAL_RCC_DAC1_IS_CLK_DISABLED

__HAL_RCC_LPDMA1_IS_CLK_DISABLED

__HAL_RCC_ADF1_IS_CLK_DISABLED

__HAL_RCC_GTZC2_IS_CLK_DISABLED

__HAL_RCC_SRAM4_IS_CLK_DISABLED

AHB Clock Source

RCC_SYSCLK_DIV1

SYSCLK not divided

RCC_SYSCLK_DIV2

SYSCLK divided by 2

RCC_SYSCLK_DIV4

SYSCLK divided by 4

RCC_SYSCLK_DIV8

SYSCLK divided by 8

RCC_SYSCLK_DIV16

SYSCLK divided by 16

RCC_SYSCLK_DIV64

SYSCLK divided by 64

RCC_SYSCLK_DIV128

SYSCLK divided by 128

RCC_SYSCLK_DIV256

SYSCLK divided by 256

RCC_SYSCLK_DIV512

SYSCLK divided by 512

APB1 APB2 APB3 Clock Source

RCC_HCLK_DIV1

HCLK not divided

RCC_HCLK_DIV2

HCLK divided by 2

RCC_HCLK_DIV4

HCLK divided by 4

RCC_HCLK_DIV8

HCLK divided by 8

RCC_HCLK_DIV16

HCLK divided by 16

APB1 Peripheral Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM5_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE

__HAL_RCC_TIM7_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_UART4_CLK_ENABLE

__HAL_RCC_UART5_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_CRS_CLK_ENABLE

__HAL_RCC_USART6_CLK_ENABLE

__HAL_RCC_I2C4_CLK_ENABLE

__HAL_RCC_LPTIM2_CLK_ENABLE

__HAL_RCC_I2C5_CLK_ENABLE

__HAL_RCC_I2C6_CLK_ENABLE

__HAL_RCC_FDCAN1_CLK_ENABLE

__HAL_RCC_UCPD_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_TIM4_CLK_DISABLE

__HAL_RCC_TIM5_CLK_DISABLE

__HAL_RCC_TIM6_CLK_DISABLE

__HAL_RCC_TIM7_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_UART4_CLK_DISABLE

__HAL_RCC_UART5_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_CRS_CLK_DISABLE

__HAL_RCC_USART6_CLK_DISABLE

__HAL_RCC_I2C4_CLK_DISABLE

__HAL_RCC_LPTIM2_CLK_DISABLE

__HAL_RCC_I2C5_CLK_DISABLE

__HAL_RCC_I2C6_CLK_DISABLE

__HAL_RCC_FDCAN1_CLK_DISABLE

__HAL_RCC_UCPD_CLK_DISABLE

APB1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_TIM4_CLK_SLEEP_ENABLE

__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

__HAL_RCC_TIM6_CLK_SLEEP_ENABLE

__HAL_RCC_TIM7_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE

__HAL_RCC_UART5_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_CRS_CLK_SLEEP_ENABLE

__HAL_RCC_USART6_CLK_SLEEP_ENABLE

__HAL_RCC_I2C4_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_ENABLE

__HAL_RCC_I2C5_CLK_SLEEP_ENABLE

__HAL_RCC_I2C6_CLK_SLEEP_ENABLE

__HAL_RCC_FDCAN1_CLK_SLEEP_ENABLE

__HAL_RCC_UCPD_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_TIM4_CLK_SLEEP_DISABLE

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE

__HAL_RCC_TIM6_CLK_SLEEP_DISABLE

__HAL_RCC_TIM7_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_DISABLE

__HAL_RCC_UART5_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_CRS_CLK_SLEEP_DISABLE

__HAL_RCC_USART6_CLK_SLEEP_DISABLE

__HAL_RCC_I2C4_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM2_CLK_SLEEP_DISABLE

__HAL_RCC_I2C5_CLK_SLEEP_DISABLE

__HAL_RCC_I2C6_CLK_SLEEP_DISABLE

__HAL_RCC_FDCAN1_CLK_SLEEP_DISABLE

__HAL_RCC_UCPD_CLK_SLEEP_DISABLE

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_TIM4_FORCE_RESET

__HAL_RCC_TIM5_FORCE_RESET

__HAL_RCC_TIM6_FORCE_RESET

__HAL_RCC_TIM7_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_UART4_FORCE_RESET

__HAL_RCC_UART5_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_CRS_FORCE_RESET

__HAL_RCC_USART6_FORCE_RESET

__HAL_RCC_I2C4_FORCE_RESET

__HAL_RCC_LPTIM2_FORCE_RESET

__HAL_RCC_I2C5_FORCE_RESET

__HAL_RCC_I2C6_FORCE_RESET

__HAL_RCC_FDCAN1_FORCE_RESET

__HAL_RCC_UCPD_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_TIM4_RELEASE_RESET

__HAL_RCC_TIM5_RELEASE_RESET

__HAL_RCC_TIM6_RELEASE_RESET

__HAL_RCC_TIM7_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_UART4_RELEASE_RESET

__HAL_RCC_UART5_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_CRS_RELEASE_RESET

__HAL_RCC_USART6_RELEASE_RESET

__HAL_RCC_I2C4_RELEASE_RESET

__HAL_RCC_LPTIM2_RELEASE_RESET

__HAL_RCC_I2C5_RELEASE_RESET

__HAL_RCC_I2C6_RELEASE_RESET

__HAL_RCC_FDCAN1_RELEASE_RESET

__HAL_RCC_UCPD_RELEASE_RESET

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_TIM4_IS_CLK_ENABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED

__HAL_RCC_TIM6_IS_CLK_ENABLED

__HAL_RCC_TIM7_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_UART4_IS_CLK_ENABLED

__HAL_RCC_UART5_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_CRS_IS_CLK_ENABLED

__HAL_RCC_USART6_IS_CLK_ENABLED

__HAL_RCC_I2C4_IS_CLK_ENABLED

__HAL_RCC_LPTIM2_IS_CLK_ENABLED

__HAL_RCC_I2C5_IS_CLK_ENABLED

__HAL_RCC_I2C6_IS_CLK_ENABLED

__HAL_RCC_FDCAN1_IS_CLK_ENABLED

__HAL_RCC_UCPD_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_TIM4_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_DISABLED

__HAL_RCC_TIM6_IS_CLK_DISABLED

__HAL_RCC_TIM7_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_UART4_IS_CLK_DISABLED

__HAL_RCC_UART5_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_CRS_IS_CLK_DISABLED

__HAL_RCC_USART6_IS_CLK_DISABLED

__HAL_RCC_I2C4_IS_CLK_DISABLED

__HAL_RCC_LPTIM2_IS_CLK_DISABLED

__HAL_RCC_I2C5_IS_CLK_DISABLED

__HAL_RCC_I2C6_IS_CLK_DISABLED

__HAL_RCC_FDCAN1_IS_CLK_DISABLED

__HAL_RCC_UCPD_IS_CLK_DISABLED

APB2 Peripheral Clock Enable Disable

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_TIM8_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_SAI1_CLK_ENABLE

__HAL_RCC_SAI2_CLK_ENABLE

__HAL_RCC_GFXTIM_CLK_ENABLE

__HAL_RCC_LTDC_CLK_ENABLE

__HAL_RCC_DSI_CLK_ENABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_TIM8_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_SAI1_CLK_DISABLE

__HAL_RCC_SAI2_CLK_DISABLE

__HAL_RCC_GFXTIM_CLK_DISABLE

__HAL_RCC_LTDC_CLK_DISABLE

__HAL_RCC_DSI_CLK_DISABLE

APB2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM8_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_SAI1_CLK_SLEEP_ENABLE

__HAL_RCC_SAI2_CLK_SLEEP_ENABLE

__HAL_RCC_GFXTIM_CLK_SLEEP_ENABLE

__HAL_RCC_LTDC_CLK_SLEEP_ENABLE

__HAL_RCC_DSI_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM8_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_SAI1_CLK_SLEEP_DISABLE

__HAL_RCC_SAI2_CLK_SLEEP_DISABLE

__HAL_RCC_GFXTIM_CLK_SLEEP_DISABLE

__HAL_RCC_LTDC_CLK_SLEEP_DISABLE

__HAL_RCC_DSI_CLK_SLEEP_DISABLE

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_TIM8_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_SAI1_FORCE_RESET

__HAL_RCC_SAI2_FORCE_RESET

__HAL_RCC_GFXTIM_FORCE_RESET

__HAL_RCC_LTDC_FORCE_RESET

__HAL_RCC_DSI_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_TIM8_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_SAI1_RELEASE_RESET

__HAL_RCC_SAI2_RELEASE_RESET

__HAL_RCC_GFXTIM_RELEASE_RESET

__HAL_RCC_LTDC_RELEASE_RESET

__HAL_RCC_DSI_RELEASE_RESET

APB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_TIM8_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_SAI1_IS_CLK_ENABLED

__HAL_RCC_SAI2_IS_CLK_ENABLED

__HAL_RCC_GFXTIM_IS_CLK_ENABLED

__HAL_RCC_LTDC_IS_CLK_ENABLED

__HAL_RCC_DSI_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_TIM8_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_TIM15_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

__HAL_RCC_SAI1_IS_CLK_DISABLED

__HAL_RCC_SAI2_IS_CLK_DISABLED

__HAL_RCC_GFXTIM_IS_CLK_DISABLED

__HAL_RCC_LTDC_IS_CLK_DISABLED

__HAL_RCC_DSI_IS_CLK_DISABLED

APB3 Peripheral Clock Enable Disable

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_I2C3_CLK_ENABLE

__HAL_RCC_LPTIM1_CLK_ENABLE

__HAL_RCC_LPTIM3_CLK_ENABLE

__HAL_RCC_LPTIM4_CLK_ENABLE

__HAL_RCC_OPAMP_CLK_ENABLE

__HAL_RCC_COMP_CLK_ENABLE

__HAL_RCC_VREF_CLK_ENABLE

__HAL_RCC_RTCAPB_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_I2C3_CLK_DISABLE

__HAL_RCC_LPTIM1_CLK_DISABLE

__HAL_RCC_LPTIM3_CLK_DISABLE

__HAL_RCC_LPTIM4_CLK_DISABLE

__HAL_RCC_OPAMP_CLK_DISABLE

__HAL_RCC_COMP_CLK_DISABLE

__HAL_RCC_VREF_CLK_DISABLE

__HAL_RCC_RTCAPB_CLK_DISABLE

APB3 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

__HAL_RCC_SPI3_CLK_SLEEP_ENABLE

__HAL_RCC_LPUART1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C3_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_ENABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_ENABLE

__HAL_RCC_OPAMP_CLK_SLEEP_ENABLE

__HAL_RCC_COMP_CLK_SLEEP_ENABLE

__HAL_RCC_VREF_CLK_SLEEP_ENABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_LPUART1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C3_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM1_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM3_CLK_SLEEP_DISABLE

__HAL_RCC_LPTIM4_CLK_SLEEP_DISABLE

__HAL_RCC_OPAMP_CLK_SLEEP_DISABLE

__HAL_RCC_COMP_CLK_SLEEP_DISABLE

__HAL_RCC_VREF_CLK_SLEEP_DISABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_DISABLE

APB3 Peripheral Force Release Reset

__HAL_RCC_APB3_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_I2C3_FORCE_RESET

__HAL_RCC_LPTIM1_FORCE_RESET

__HAL_RCC_LPTIM3_FORCE_RESET

__HAL_RCC_LPTIM4_FORCE_RESET

__HAL_RCC_OPAMP_FORCE_RESET

__HAL_RCC_COMP_FORCE_RESET

__HAL_RCC_VREF_FORCE_RESET

__HAL_RCC_APB3_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_I2C3_RELEASE_RESET

__HAL_RCC_LPTIM1_RELEASE_RESET

__HAL_RCC_LPTIM3_RELEASE_RESET

__HAL_RCC_LPTIM4_RELEASE_RESET

__HAL_RCC_OPAMP_RELEASE_RESET

__HAL_RCC_COMP_RELEASE_RESET

__HAL_RCC_VREF_RELEASE_RESET

APB3 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_I2C3_IS_CLK_ENABLED

__HAL_RCC_LPTIM1_IS_CLK_ENABLED

__HAL_RCC_LPTIM3_IS_CLK_ENABLED

__HAL_RCC_LPTIM4_IS_CLK_ENABLED

__HAL_RCC_OPAMP_IS_CLK_ENABLED

__HAL_RCC_COMP_IS_CLK_ENABLED

__HAL_RCC_VREF_IS_CLK_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_I2C3_IS_CLK_DISABLED

__HAL_RCC_LPTIM1_IS_CLK_DISABLED

__HAL_RCC_LPTIM3_IS_CLK_DISABLED

__HAL_RCC_LPTIM4_IS_CLK_DISABLED

__HAL_RCC_OPAMP_IS_CLK_DISABLED

__HAL_RCC_COMP_IS_CLK_DISABLED

__HAL_RCC_VREF_IS_CLK_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_DISABLED

RCC attributes

RCC_NSEC_PRIV

Non-secure Privilege attribute item

RCC_NSEC_NPRIV

Non-secure Non-privilege attribute item

RCC_SEC_PRIV

Secure Privilege attribute item

RCC_SEC_NPRIV

Secure Non-privilege attribute item

RCC Backup Domain Reset

__HAL_RCC_BACKUPRESET_FORCE

Description:

- Macros to force or release the Backup domain reset.

Return value:

- None

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

__HAL_RCC_BACKUPRESET_RELEASE

RCC Exported Constants

HSE_TIMEOUT_VALUE

HSI_TIMEOUT_VALUE

MSI_TIMEOUT_VALUE

CR_REG_INDEX

BDCR_REG_INDEX

CSR_REG_INDEX

CRRCR_REG_INDEX

RCC_FLAG_MASK

RCC Exported Macros

__HAL_RCC_HSI_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 16MHz oscillator (HSI).

Return value:

- None

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. This parameter can be: ENABLE or DISABLE. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed 16MHz oscillator (HSI) calibration value.

Parameters:

- __HSICALIBRATIONVALUE__: specifies the calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between 0 and 0x20.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

__HAL_RCC_HSISTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal High Speed oscillator (HSI) in STOP mode to be quickly available as kernel clock for USARTs, LPUART and I2Cs.

Return value:

- None

Notes:

- Keeping the HSI ON in STOP mode allows to avoid slowing down the communication speed because of the HSI startup time. The enable of this function has not effect on the HSION bit. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSISTOP_DISABLE

__HAL_RCC_MSIKSTOP_ENABLE

Description:

- Macros to enable or disable the force of the Internal Multi Speed kernel oscillator (MSIK) in STOP mode to be quickly available as kernel clock for USARTs, LPUART and I2Cs.

Return value:

- None

Notes:

- Keeping the MSIK ON in STOP mode allows to avoid slowing down the communication speed because of the MSIK startup time. The enable of this function has not effect on the MSIKON bit. The MSIKERON must be configured at 0 before entreing stop 3 mode. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_MSIKSTOP_DISABLE

__HAL_RCC_MSI_ENABLE

Description:

- Macros to enable or disable the Internal Multi Speed oscillator (MSI).

Return value:

- None

Notes:

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. After enabling the MSI, the application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

__HAL_RCC_MSI_DISABLE

__HAL_RCC_MSI_RANGE_CONFIG

Description:

- Macro configures the Internal Multi Speed oscillator (MSI) clock range in run mode.

Parameters:

- __MSIRANGEVALUE__: specifies the MSI clock range. This parameter must be one of the following values:
  - RCC_MSIRANGE_0 MSI clock is around 48 MHz
  - RCC_MSIRANGE_1 MSI clock is around 24 KHz
  - RCC_MSIRANGE_2 MSI clock is around 16 MHz
  - RCC_MSIRANGE_3 MSI clock is around 12 MHz
  - RCC_MSIRANGE_4 MSI clock is around 4 MHz (default after Reset)
  - RCC_MSIRANGE_5 MSI clock is around 2 MHz
  - RCC_MSIRANGE_6 MSI clock is around 1.33 MHz
  - RCC_MSIRANGE_7 MSI clock is around 1 MHz
  - RCC_MSIRANGE_8 MSI clock is around 3.072 MHz
  - RCC_MSIRANGE_9 MSI clock is around 1.536 MHz
  - RCC_MSIRANGE_10 MSI clock is around 1.024 MHz
  - RCC_MSIRANGE_11 MSI clock is around 768 KHz
  - RCC_MSIRANGE_12 MSI clock is around 400 KHz
  - RCC_MSIRANGE_13 MSI clock is around 200 KHz
  - RCC_MSIRANGE_14 MSI clock is around 133 KHz
  - RCC_MSIRANGE_15 MSI clock is around 100 KHz

Return value:

- None

Notes:

- After restart from Reset , the MSI clock is around 4 MHz. After stop the startup clock can be MSI (at any of its possible frequencies, the one that was used before entering stop mode) or HSI. After Standby its frequency can be selected between 4 possible values (1, 3.072, 4 or 8 MHz). MSIRANGE can be modified when MSI is OFF (MSION=0) or when MSI is ready (MSIRDY=1). The MSI clock range after reset can be modified on the fly.

__HAL_RCC_MSIK_RANGE_CONFIG

Description:

- Macro configures the Internal Multi Speed kernel oscillator (MSIK) clock range in run mode.

Parameters:

- __MSIKRANGEVALUE__: specifies the MSI clock range.
  – RCC_MSIKRANGE_0 MSIK clock is around 48 MHz
  – RCC_MSIKRANGE_1 MSIK clock is around 24 KHz
  – RCC_MSIKRANGE_2 MSIK clock is around 16 MHz
  – RCC_MSIKRANGE_3 MSIK clock is around 12 MHz
  – RCC_MSIKRANGE_4 MSIK clock is around 4 MHz (default after Reset)
  – RCC_MSIKRANGE_5 MSIK clock is around 2 MHz
  – RCC_MSIKRANGE_6 MSIK clock is around 1.33 MHz
  – RCC_MSIKRANGE_7 MSIK clock is around 1 MHz
  – RCC_MSIKRANGE_8 MSIK clock is around 3.072 MHz
  – RCC_MSIKRANGE_9 MSIK clock is around 1.536 MHz
  – RCC_MSIKRANGE_10 MSIK clock is around 1.024 MHz
  – RCC_MSIKRANGE_11 MSIK clock is around 768 KHz
  – RCC_MSIKRANGE_12 MSIK clock is around 400 KHz
  – RCC_MSIKRANGE_13 MSIK clock is around 200 KHz
  – RCC_MSIKRANGE_14 MSIK clock is around 133 KHz
  – RCC_MSIKRANGE_15 MSIK clock is around 100 KHz

Return value:

- None

Notes:

- After restart from Reset , the MSIK clock is around 4 MHz. After stop the startup clock can be MSIK (at any of its possible frequencies, the one that was used before entering stop mode) or HSI. After Standby its frequency can be selected between 4 possible values (1, 3.072, 4 or 8 MHz). MSIKRANGE can be modified when MSIK is OFF (MSIKON=0) or when MSIK is ready (MSIKRDY=1). The MSI clock range after reset can be modified on the fly.

__HAL_RCC_MSIBIAS_SELECTION_ENABLE

Description:

- Macros to enable or disable the MSI bias mode selection.

Return value:

- None

Notes:

- By default the MSI bias is in continuous mode in order to maintain the output clocks accuracy. Setting this bit reduces the MSI consumption under range 4 but decrease its accuracy.

__HAL_RCC_MSIBIAS_SELECTION_DISABLE

__HAL_RCC_LSE_GLITCHFILTER_ENABLE

Description:

- Macros to enable or disable LSE clock glitch filter .

Return value:

- None

Notes:

- The glitches on LSE can be filtred by setting the LSEGFON. LSEGFON must be written when the LSE is disabled (LSEON = 0 and LSERDY = 0).

__HAL_RCC_LSE_GLITCHFILTER_DISABLE

__HAL_RCC_MSI_STANDBY_RANGE_CONFIG

Description:

- Macro configures the Internal Multi Speed oscillator (MSI) clock range after Standby mode After Standby its frequency can be selected between 5 possible values (4, 2, 1.33, 1, or 3.072 MHz).

Parameters:

- __MSIRANGEVALUE__: specifies the MSI clock range. This parameter must be one of the following values:
  – RCC_MSIRANGE_4 MSI clock is around 4 MHz (default after Reset)
  – RCC_MSIRANGE_5 MSI clock is around 2 MHz
  – RCC_MSIRANGE_6 MSI clock is around 1.33 MHz
  – RCC_MSIRANGE_7 MSI clock is around 1 MHz
  – RCC_MSIRANGE_8 MSI clock is around 3.072 MHz

Return value:

- None

__HAL_RCC_MSIK_STANDBY_RANGE_CONFIG

Description:

- Macro configures the Internal Multi Speed oscillator (MSIK) clock range after Standby mode After Standby its frequency can be selected between 5 possible values (4, 2, 1.33, 1, or 3.072 MHz).

Parameters:

- __MSIKRANGEVALUE__: specifies the MSIK clock range. This parameter must be one of the following values:
  – RCC_MSIKRANGE_4 MSIK clock is around 4 MHz (default after Reset)
  – RCC_MSIKRANGE_5 MSIK clock is around 2 MHz
  – RCC_MSIKRANGE_6 MSIK clock is around 1.33 MHz
  – RCC_MSIKRANGE_7 MSIK clock is around 1 MHz
  – RCC_MSIKRANGE_8 MSIK clock is around 3.072 MHz

Return value:

- None

__HAL_RCC_GET_MSI_RANGE

Description:

- Macro to get the Internal Multi Speed oscillator (MSI) clock range in run mode.

Return value:

- MSI: clock range. This parameter must be one of the following values:
  – RCC_MSIRANGE_0 MSI clock is around 48 MHz
  – RCC_MSIRANGE_1 MSI clock is around 24 KHz
  – RCC_MSIRANGE_2 MSI clock is around 16 MHz
  – RCC_MSIRANGE_3 MSI clock is around 12 MHz
  – RCC_MSIRANGE_4 MSI clock is around 4 MHz
  – RCC_MSIRANGE_5 MSI clock is around 2 MHz
  – RCC_MSIRANGE_6 MSI clock is around 1.33 MHz
  – RCC_MSIRANGE_7 MSI clock is around 1 MHz
  – RCC_MSIRANGE_8 MSI clock is around 3.072 MHz
  – RCC_MSIRANGE_9 MSI clock is around 1.536 MHz
  – RCC_MSIRANGE_10 MSI clock is around 1.024 MHz
  – RCC_MSIRANGE_11 MSI clock is around 768 KHz
  – RCC_MSIRANGE_12 MSI clock is around 400 KHz
  – RCC_MSIRANGE_13 MSI clock is around 200 KHz
  – RCC_MSIRANGE_14 MSI clock is around 133 KHz
  – RCC_MSIRANGE_15 MSI clock is around 100 KHz

__HAL_RCC_GET_MSIK_RANGE

Description:

- Macro to get the Internal Multi Speed kernel oscillator (MSIK) clock range in run mode.

Return value:

- MSIK: clock range. This parameter must be one of the following values:
  – RCC_MSIRANGE_0 MSI clock is around 48 MHz
  – RCC_MSIRANGE_1 MSI clock is around 24 KHz
  – RCC_MSIRANGE_2 MSI clock is around 16 MHz
  – RCC_MSIRANGE_3 MSI clock is around 12 MHz
  – RCC_MSIRANGE_4 MSI clock is around 4 MHz
  – RCC_MSIRANGE_5 MSI clock is around 2 MHz
  – RCC_MSIRANGE_6 MSI clock is around 1.33 MHz
  – RCC_MSIRANGE_7 MSI clock is around 1 MHz
  – RCC_MSIRANGE_8 MSI clock is around 3.072 MHz
  – RCC_MSIRANGE_9 MSI clock is around 1.536 MHz
  – RCC_MSIRANGE_10 MSI clock is around 1.024 MHz
  – RCC_MSIRANGE_11 MSI clock is around 768 KHz
  – RCC_MSIRANGE_12 MSI clock is around 400 KHz
  – RCC_MSIRANGE_13 MSI clock is around 200 KHz
  – RCC_MSIRANGE_14 MSI clock is around 133 KHz
  – RCC_MSIRANGE_15 MSI clock is around 100 KHz

__HAL_RCC_LSI_ENABLE

Description:

- Macros to enable or disable the Internal Low Speed oscillator (LSI).

Return value:

- None

Notes:

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_HSE_CONFIG

Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- __STATE__: specifies the new state of the HSE. This parameter can be one of the following values:
  – RCC_HSE_OFF Turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
  – RCC_HSE_ON Turn ON the HSE oscillator.
  – RCC_HSE_BYPASS HSE oscillator bypassed with external clock.
  – RCC_HSE_BYPASS_DIGITAL HSE oscillator bypassed through I/O Schmitt trigger .

Return value:

- None

Notes:

- Transition HSE Bypass to HSE On and HSE On to HSE Bypass are not supported by this macro. User should request a transition to HSE Off first and then HSE On or HSE Bypass. After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable it). The HSE is stopped by hardware when entering STOP and STANDBY or shutdown modes.

__HAL_RCC_LSESYS_ENABLE

Description:

- Macro to enable or disable the LSE system clock.

Return value:

- None

Notes:

- This clock can be used by any peripheral when its source clock is the LSE or at system in case of one of the LSCOSEL, MCO, MSI PLL mode or CSS on LSE is needed. The LSESYS clock can be generated even if LSESYSEN= 0 if the LSE clock is requested by the CSS on LSE, by a peripheral or any other source clock using LSE.

__HAL_RCC_LSESYS_DISABLE

__HAL_RCC_LSI_DIV_CONFIG

Description:

- Macro to set Low-speed clock (LSI) divider.

Parameters:

- __DIVIDER__: : specifies the divider value This parameter can be one of the following values
  – RCC_LSI_DIV1
  – RCC_LSI_DIV128

Return value:

- None

Notes:

- This bit can be written only when the LSI is disabled (LSION = 0 and LSIRDY = 0). The LSIPREDIV cannot be changed if the LSI is used by the IWDG or by the RTC.

__HAL_RCC_LSE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__: specifies the new state of the LSE. This parameter can be one of the following values:
  - RCC_LSE_OFF Turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
  - RCC_LSE_ON_RTC_ONLY Turn ON the LSE oscillator to be used only for RTC.
  - RCC_LSE_ON Turn ON the LSE oscillator to be used by any peripheral.
  - RCC_LSE_BYPASS_RTC_ONLY LSE oscillator bypassed with external clock to be used only for RTC.
  - RCC_LSE_BYPASS LSE oscillator bypassed with external clock to be used by any peripheral

Return value:

- None

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. User should request a transition to LSE Off first and then LSE On or LSE Bypass. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

__HAL_RCC_HSI48_ENABLE

Description:

- Macros to enable or disable the Internal High Speed 48MHz oscillator (HSI48).

Return value:

- None

Notes:

- The HSI48 is stopped by hardware when entering STOP and STANDBY modes. After enabling the HSI48, the application software should wait on HSI48RDY flag to be set indicating that HSI48 clock is stable. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_HSI48_DISABLE

__HAL_RCC_MSIK_ENABLE

Description:

- Macros to enable or disable the Internal multi-speed RC oscillator clock (MSIK).

Return value:

- None

Notes:

- if the peripheral requests its kernel clock in Stop 0 or Stop 1 mode,MSIK is woken up After enabling the MSIK, the application software should wait on MSIKRDY flag to be set indicating that MSIK clock is stable. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_MSIK_DISABLE

__HAL_RCC_SHSI_ENABLE

Description:

- Macros to enable or disable the secure Internal High Speed oscillator (SHSI).

Return value:

- None

Notes:

- The SHSI is stopped by hardware when entering STOP and STANDBY modes. After enabling the SHSI, the application software should wait on SHSI flag to be set indicating that SHSI clock is stable. This parameter can be: ENABLE or DISABLE.

__HAL_RCC_SHSI_DISABLE

__HAL_RCC_RTC_CONFIG

Description:

- Macros to configure the RTC clock (RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock source. This parameter can be one of the following values:
  – RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock.
  – RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  – RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  – RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32 selected

Return value:

- None

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it cannot be changed unless the Backup domain is reset using __HAL_RCC_BACKUPRESET_FORCE() macro, or by a Power On Reset (POR).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

__HAL_RCC_GET_RTC_SOURCE

Description:

- Macro to get the RTC clock source.

Return value:

- The: returned value can be one of the following:
  – RCC_RTCCLKSOURCE_NO_CLK No clock selected as RTC clock.
  – RCC_RTCCLKSOURCE_LSE LSE selected as RTC clock.
  – RCC_RTCCLKSOURCE_LSI LSI selected as RTC clock.
  – RCC_RTCCLKSOURCE_HSE_DIV32 HSE clock divided by 32 selected

__HAL_RCC_PLL_ENABLE

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL can not be disabled if it is used as system clock source The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

__HAL_RCC_PLLCLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL_P_CLK, PLL_Q_CLK, PLL_R_CLK)

Parameters:

- __PLL1_CLOCKOUT__: specifies the PLL clock to be outputted This parameter can be one of the following values:
  - RCC_PLL1_DIVP: This clock is used to generate an accurate clock to achieve, high-quality audio performance on SAI interface.
  - RCC_PLL1_DIVQ: This Clock is used to generate the clock for the USB FS(48 MHz), the random analog generator (<=48 MHz) and the OCTOSPI1/2.
  - RCC_PLL1_DIVR: This Clock is used to generate the high speed system clock (up to 160MHz)

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL, This is mainly used to save Power.

__HAL_RCC_PLLCLKOUT_DISABLE

__HAL_RCC_GET_PLLCLKOUT_CONFIG

Description:

- Macro to get the PLL clock output enable status.

Parameters:

- __PLL1_CLOCKOUT__: specifies the PLL1 clock to be output. This parameter can be one of the following values:
  - RCC_PLL1_DIVP: This clock is used to generate an accurate clock to achieve, high-quality audio performance on SAI interface.
  - RCC_PLL1_DIVQ: This Clock is used to generate the clock for the USB FS(48 MHz), the random analog generator (<=48 MHz) and the OCTOSPI1/2.
  - RCC_PLL1_DIVR: This Clock is used to generate the high speed system clock (up to 160MHz)

Return value:

- SET: / RESET

__HAL_RCC_PLL_FRACN_ENABLE

Description:

- Enables or disables Fractional Part Of The Multiplication Factor of PLL1 VCO.

Return value:

- None

Notes:

- Enabling/disabling Fractional Part can be any time without the need to stop the PLL1

__HAL_RCC_PLL_FRACN_DISABLE

__HAL_RCC_PLL_CONFIG

Description:

- Macro to configure the main PLL clock source, multiplication and division factors.

Parameters:

- __PLL1SOURCE__: specifies the PLL entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_MSI: MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry
- __PLL1MBOOST__: specifies the division factor for the EPOD clock This parameter must be a value of
- __PLL1M__: specifies the division factor for PLL VCO input clock This parameter must be a number between 1 and 63.
- __PLL1N__: specifies the multiplication factor for PLL VCO output clock This parameter must be a number between 4 and 512.
- __PLL1P__: specifies the division factor for peripheral kernel clocks. This parameter must be a number between 1 and 128
- __PLL1Q__: specifies the division factor for peripheral kernel clocks. This parameter must be a number between 1 and 128
- __PLL1R__: specifies the division factor for system clock. This parameter must be a number between 1 and 128 (Only division by 1 and even division are allowed)

Return value:

- None

Notes:

- This function must be used only when the main PLL is disabled.
- This clock source (__PLL1SOURCE__) is common for the main PLL1 (main PLL) and PLL2 & PLL3 .
- You have to set the PLLM parameter correctly to ensure that the VCO input frequency ranges from 1 to 16 MHz.
- You have to set the PLLN parameter correctly to ensure that the VCO output frequency is between 128 and 544 MHz(Voltage range 1 or 2) between 128 and 330 MHZ (Voltage range 3) and not allowed for Voltage range 4.

__HAL_RCC_PLL_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLLs clock source.

Parameters:

- __PLL1SOURCE__: specifies the PLLs entry clock source. This parameter can be one of the following values:
  - RCC_PLLSOURCE_MSI: MSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL clock entry
  - RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL clock entry

Notes:

- This function must be used only when all PLLs are disabled.

__HAL_RCC_PLL_FRACN_CONFIG

Description:

- Macro to configure the main PLL clock Fractional Part Of The Multiplication Factor.

Parameters:

- __PLL1FRACN__: specifies Fractional Part Of The Multiplication Factor for PLL1 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL1 VCO
- Warning: The software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL1VCOSEL = 0 150 to 420 MHz if PLL1VCOSEL = 1.

__HAL_RCC_PLL_VCIRANGE

Description:

- Macro to select the PLL1 reference frequency range.

Parameters:

- __PLL1VCIRange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  - RCC_PLLVCIRANGE_0: Range frequency is between 4 and 8 MHz
  - RCC_PLLVCIRANGE_1: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_GET_PLL_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL1 clock source.

Return value:

- The: oscillator used as PLL1 clock source. The returned value can be one of the following:
  - RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_MSI: MSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  - RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_SYSCLK_CONFIG

Description:

- Macro to configure the system clock source.

Parameters:

- __SYSCLKSOURCE__: specifies the system clock source. This parameter can be one of the following values:
  - RCC_SYSCLKSOURCE_MSI: MSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_PLL1CLK: PLL1 output is used as system clock source.

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_STATUS_MSI: MSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_PLL1CLK: PLL1 used as system clock.

__HAL_RCC_LSEDRIVE_CONFIG

Description:

- Macro to configure the External Low Speed oscillator (LSE) drive capability.

Parameters:

- __LSEDRIVE__: specifies the new state of the LSE drive capability. This parameter can be one of the following values:
  - RCC_LSEDRIVE_LOW LSE oscillator low drive capability.
  - RCC_LSEDRIVE_MEDIUMLOW LSE oscillator medium low drive capability.
  - RCC_LSEDRIVE_MEDIUMHIGH LSE oscillator medium high drive capability.
  - RCC_LSEDRIVE_HIGH LSE oscillator high drive capability.

Return value:

- None

Notes:

- As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). The LSE drive can be decreased to the lower drive capability (LSEDRV = 0) when the LSE is ON. However, once LSEDRV is selected, the drive capability can not be increased if LSEON = 1.

__HAL_RCC_WAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the wake up from stop clock.

Parameters:

- __STOPWUCLK__: specifies the clock source used after wake up from stop. This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_MSI MSI selected as system clock source and CSS backup clock
  - RCC_STOP_WAKEUPCLOCK_HSI HSI selected as system clock source and CSS backup clock

Return value:

- None

Notes:

- The selected clock is also used as emergency clock for the clock security system on HSE.

__HAL_RCC_KERWAKEUPSTOP_CLK_CONFIG

Description:

- Macro to configure the Kernel wake up from stop clock.

Parameters:

- __RCC_STOPKERWUCLK__: specifies the Kernel clock source used after wake up from stop This parameter can be one of the following values:
  - RCC_STOP_KERWAKEUPCLOCK_MSI: MSI selected as Kernel clock source
  - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI selected as Kernel clock source

Return value:

- None

__HAL_RCC_MCO1_CONFIG

Description:

- Macro to configure the MCO clock.

Parameters:

- __MCOCLKSOURCE__: specifies the MCO clock source. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK MCO output disabled
  - RCC_MCO1SOURCE_SYSCLK System clock selected as MCO source
  - RCC_MCO1SOURCE_MSI MSI clock selected as MCO source
  - RCC_MCO1SOURCE_HSI HSI clock selected as MCO source
  - RCC_MCO1SOURCE_HSE HSE clock selected as MCO source
  - RCC_MCO1SOURCE_PLL1CLK Main PLL clock selected as MCO source
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO source
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO source
  - RCC_MCO1SOURCE_HSI48 HSI48 clock selected as MCO source for devices with HSI48
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCODIV_1 MCO clock source is divided by 1
  - RCC_MCODIV_2 MCO clock source is divided by 2
  - RCC_MCODIV_4 MCO clock source is divided by 4
  - RCC_MCODIV_8 MCO clock source is divided by 8
  - RCC_MCODIV_16 MCO clock source is divided by 16

IS_RCC_PLLSOURCE

IS_RCC_PLLM_VALUE

IS_RCC_PLLN_VALUE

IS_RCC_PLLP_VALUE

IS_RCC_PLLQ_VALUE

IS_RCC_PLLR_VALUE

IS_RCC_STOP_WAKEUPCLOCK

IS_RCC_STOP_KERWAKEUPCLOCK

IS_RCC_MSI_CLOCK_RANGE

IS_RCC_MSI_STANDBY_CLOCK_RANGE

IS_RCC_MSIK_CLOCK_RANGE

IS_RCC_MSIK_STANDBY_CLOCK_RANGE

Flags

RCC_FLAG_MSIRDY
MSI Ready flag

RCC_FLAG_MSIKRDY
MSI Ready flag

RCC_FLAG_HSIRDY
HSI Ready flag

RCC_FLAG_HSERDY
HSE Ready flag

RCC_FLAG_PLL1RDY
PLL Ready flag

RCC_FLAG_PLL2RDY
PLL2 Ready flag

RCC_FLAG_PLL3RDY
PLL3 Ready flag

RCC_FLAG_SHSIRDY
SHSI Ready flag

RCC_FLAG_HSI48RDY
HSI48 Ready flag

RCC_FLAG_LSERDY
LSE Ready flag

RCC_FLAG_LSESYSRDY
LSESYS Ready flag

RCC_FLAG_LSECSSD
LSE Clock Security System Interrupt flag

RCC_FLAG_LSIRDY
LSI Ready flag

RCC_FLAG_RMVF
Remove reset flag

RCC_FLAG_OBLRST
Option Byte Loader reset flag

RCC_FLAG_PINRST
PIN reset flag

RCC_FLAG_BORRST
BOR reset flag

RCC_FLAG_SFTRST

Software Reset flag

RCC_FLAG_IWDGRST

Independent Watchdog reset flag

RCC_FLAG_WWDGRST

Window watchdog reset flag

RCC_FLAG_LPWRRST

Low-Power reset flag

Flags Interrupts Management

__HAL_RCC_ENABLE_IT

Description:
- Enable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to enable the selected interrupts).

Parameters:
- __INTERRUPT__: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_MSIRDY HSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:
- None

__HAL_RCC_DISABLE_IT

Description:
- Disable RCC interrupt (Perform Byte access to RCC_CIR[14:8] bits to disable the selected interrupts).

Parameters:
- __INTERRUPT__: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_MSIRDY HSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt

Return value:
- None

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits (Perform Byte access to RCC_CIR[23:16] bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt
  - RCC_IT_MSIKRDY MSIK ready interrupt
  - RCC_IT_SHSIRDY SHSI ready interrupt

Return value:

- None

__HAL_RCC_GET_IT

Description:

- Check whether the RCC interrupt has occurred or not.

Parameters:

- __INTERRUPT__: specifies the RCC interrupt source to check. This parameter can be one of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt
  - RCC_IT_LSERDY LSE ready interrupt
  - RCC_IT_MSIRDY MSI ready interrupt
  - RCC_IT_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_PLLRDY Main PLL ready interrupt
  - RCC_IT_PLL2RDY PLL2 ready interrupt
  - RCC_IT_PLL3RDY PLL3 ready interrupt
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt
  - RCC_IT_MSIKRDY MSIK ready interrupt
  - RCC_IT_SHSIRDY SHSI ready interrupt

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

Description:

- Set RMVF bit to clear the reset flags.

Return value:

- None

__HAL_RCC_GET_FLAG

Description:

- Check whether the selected RCC flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - RCC_FLAG_MSIRDY MSI oscillator clock ready
  - RCC_FLAG_HSIRDY HSI oscillator clock ready
  - RCC_FLAG_HSERDY HSE oscillator clock ready
  - RCC_FLAG_PLL1RDY Main PLL1 clock ready
  - RCC_FLAG_PLL2RDY PLL2 clock ready
  - RCC_FLAG_PLL3RDY PLL3 clock ready
  - RCC_FLAG_HSI48RDY HSI48 clock ready
  - RCC_FLAG_LSERDY LSE oscillator clock ready
  - RCC_FLAG_LSECSSD Clock security system failure on LSE oscillator detection
  - RCC_FLAG_LSIRDY LSI oscillator clock ready
  - RCC_FLAG_BORRST BOR reset
  - RCC_FLAG_OBLRST OBLRST reset
  - RCC_FLAG_PINRST Pin reset
  - RCC_FLAG_RMVF Remove reset Flag
  - RCC_FLAG_SFTRST Software reset
  - RCC_FLAG_IWDGRST Independent Watchdog reset
  - RCC_FLAG_WWDGRST Window Watchdog reset
  - RCC_FLAG_LPWRRST Low Power reset

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

RCC_HSE_BYPASS

External clock source for HSE clock

RCC_HSE_BYPASS_DIGITAL

HSI48 Config

RCC_HSI48_OFF

HSI48 clock deactivation

RCC_HSI48_ON

HSI48 clock activation

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSICALIBRATION_DEFAULT

Interrupts

RCC_IT_LSIRDY

LSI Ready Interrupt flag

RCC_IT_LSERDY

LSE Ready Interrupt flag

RCC_IT_MSIRDY

MSI Ready Interrupt flag

RCC_IT_HSIRDY

HSI16 Ready Interrupt flag

RCC_IT_HSERDY

HSE Ready Interrupt flag

RCC_IT_HSI48RDY

HSI48 Ready Interrupt flag

RCC_IT_PLLRDY

PLL1 Ready Interrupt flag

RCC_IT_PLL2RDY

PLL2 Ready Interrupt flag

RCC_IT_PLL3RDY

PLL3 Ready Interrupt flag

RCC_IT_CSS

Clock Security System Interrupt flag

RCC_IT_MSIKRDY

MSIK Ready Interrupt flag

RCC_IT_SHSIRDY

SHSI Ready Interrupt flag

RCC items

RCC_HSI

RCC_HSE

RCC_MSI

RCC_LSI

RCC_LSE

RCC_SYSCLK

RCC_PRESC

RCC_PLL1

RCC_PLL2

RCC_PLL3

RCC_ICLK

RCC_HSI48

RCC_RMVF

RCC_ALL

LSE Drive Config

RCC_LSEDRIVE_LOW

LSE low drive capability

RCC_LSEDRIVE_MEDIUMLOW

LSE medium low drive capability

RCC_LSEDRIVE_MEDIUMHIGH

LSE medium high drive capability

RCC_LSEDRIVE_HIGH

LSE high drive capability

LSE Config

RCC_LSE_OFF

LSE clock deactivation

RCC_LSE_ON_RTC_ONLY

LSE clock activation for RTC only

RCC_LSE_ON

LSE clock activation for RCC and peripherals

RCC_LSE_BYPASS_RTC_ONLY

External clock source for LSE clock

RCC_LSE_BYPASS

External clock source for LSE clock

LSI Config

RCC_LSI_OFF

LSI clock deactivation

RCC_LSI_ON

LSI clock activation

LSI Div

RCC_LSI_DIV1

LSI clock is not divided

RCC_LSI_DIV128

LSI clock is divided by 128

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

MCO1 output disabled, no clock on MCO1

RCC_MCO1SOURCE_SYSCLK

SYSCLK selection as MCO1 source

RCC_MCO1SOURCE_MSI

MSI selection as MCO1 source

RCC_MCO1SOURCE_HSI

HSI selection as MCO1 source

RCC_MCO1SOURCE_HSE

HSE selection as MCO1 source

RCC_MCO1SOURCE_PLL1CLK

PLL1CLK selection as MCO1 source

RCC_MCO1SOURCE_LSI

LSI selection as MCO1 source

RCC_MCO1SOURCE_LSE

LSE selection as MCO1 source

RCC_MCO1SOURCE_HSI48

HSI48 selection as MCO1 source

RCC_MCO1SOURCE_MSIK

MSIK selection as MCO1 source

MCO1 Clock Prescaler

RCC_MCODIV_1

MCO is divided by 1

RCC_MCODIV_2

MCO is divided by 2

RCC_MCODIV_4

MCO is divided by 4

RCC_MCODIV_8

MCO is divided by 8

RCC_MCODIV_16

MCO is divided by 16

MCO Index

RCC_MCO1

RCC_MCO

MCO1 to be compliant with other families with 2 MCOs

MSIK Clock Range

RCC_MSIKRANGE_0

MSIK = 48 MHz

RCC_MSIKRANGE_1

MSIK = 24 MHz

RCC_MSIKRANGE_2

MSIK = 16 MHz

RCC_MSIKRANGE_3

MSIK = 12 MHz

RCC_MSIKRANGE_4

MSIK = 4 MHz

RCC_MSIKRANGE_5

MSIK = 2 MHz

RCC_MSIKRANGE_6

MSIK = 1.33 MHz

RCC_MSIKRANGE_7

MSIK = 1 MHz

RCC_MSIKRANGE_8

MSIK = 3.072 MHz

RCC_MSIKRANGE_9

MSIK = 1.536 MHz

RCC_MSIKRANGE_10

MSIK = 1.024 MHz

RCC_MSIKRANGE_11

MSIK = 768 KHz

RCC_MSIKRANGE_12

MSIK = 400 KHz

RCC_MSIKRANGE_13

MSIK = 200 KHz

RCC_MSIKRANGE_14

MSIK = 133 KHz

RCC_MSIKRANGE_15

MSIK = 100 KHz

MSIK Config

RCC_MSIK_OFF

MSIK clock deactivation

RCC_MSIK_ON

MSIK clock activation

MSI Clock Range

RCC_MSIRANGE_0

MSI = 48 MHz

RCC_MSIRANGE_1

MSI = 24 MHz

RCC_MSIRANGE_2

MSI = 16 MHz

RCC_MSIRANGE_3

MSI = 12 MHz

RCC_MSIRANGE_4

MSI = 4 MHz

RCC_MSIRANGE_5

MSI = 2 MHz

RCC_MSIRANGE_6

MSI = 1.33 MHz

RCC_MSIRANGE_7

MSI = 1 MHz

RCC_MSIRANGE_8

MSI = 3.072 MHz

RCC_MSIRANGE_9

MSI = 1.536 MHz

RCC_MSIRANGE_10

MSI = 1.024 MHz

RCC_MSIRANGE_11

MSI = 768 KHz

RCC_MSIRANGE_12

MSI = 400 KHz

RCC_MSIRANGE_13

MSI = 200 KHz

RCC_MSIRANGE_14

MSI = 133 KHz

RCC_MSIRANGE_15

MSI = 100 KHz

MSI Config

RCC_MSI_OFF

MSI clock deactivation

RCC_MSI_ON

MSI clock activation

RCC_MSICALIBRATION_DEFAULT

Default MSI calibration trimming value

Oscillator Type

RCC_OSCILLATORTYPE_NONE

Oscillator configuration unchanged

RCC_OSCILLATORTYPE_HSE

HSE to configure

RCC_OSCILLATORTYPE_HSI

HSI to configure

RCC_OSCILLATORTYPE_LSE

LSE to configure

RCC_OSCILLATORTYPE_LSI

LSI to configure

RCC_OSCILLATORTYPE_MSI

MSI to configure

RCC_OSCILLATORTYPE_HSI48

HSI48 to configure

RCC_OSCILLATORTYPE_MSIK

MSIK to configure

RCC_OSCILLATORTYPE_SHSI

SHSI to configure

RCC_OSCILLATORTYPE_ALL

All Oscillator to configure

RCC PLL2 Clock Output

RCC_PLL2_DIVP

RCC_PLL2_DIVQ

RCC_PLL2_DIVR

RCC PLL3 Clock Output

RCC_PLL3_DIVP

RCC_PLL3_DIVQ

RCC_PLL3_DIVR

PLLMBOOST EPOD Clock Divider

RCC_PLLMBOOST_DIV1

RCC_PLLMBOOST_DIV2

RCC_PLLMBOOST_DIV4

RCC_PLLMBOOST_DIV6

RCC_PLLMBOOST_DIV8

RCC_PLLMBOOST_DIV10

RCC_PLLMBOOST_DIV12

RCC_PLLMBOOST_DIV14

RCC_PLLMBOOST_DIV16

RCC PLL Clock Output

RCC_PLL1_DIVP

RCC_PLL1_DIVQ

RCC_PLL1_DIVR

RCC PLL Clock Source

RCC_PLLSOURCE_NONE

RCC_PLLSOURCE_MSI

RCC_PLLSOURCE_HSI

RCC_PLLSOURCE_HSE

RCC PLL Config

RCC_PLL_NONE

RCC_PLL_OFF

RCC_PLL_ON

RCC PLL1 VCI Range

RCC_PLLVCIRANGE_0

RCC_PLLVCIRANGE_1

Reset Flag

RCC_RESET_FLAG_OBL

Option Byte Loader reset flag

RCC_RESET_FLAG_PIN

PIN reset flag

RCC_RESET_FLAG_PWR

BOR or POR/PDR reset flag

RCC_RESET_FLAG_SW

Software Reset flag

RCC_RESET_FLAG_IWDG

Independent Watchdog reset flag

RCC_RESET_FLAG_WWDG

Window watchdog reset flag

RCC_RESET_FLAG_LPWR

Low power reset flag

RCC_RESET_FLAG_ALL

RCC RTC Clock Configuration

__HAL_RCC_RTC_ENABLE

Description:

- Macros to enable or disable the RTC clock.

Return value:

- None

Notes:

- As the RTC is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the RTC (to be done once after reset). These macros must be used after the RTC clock source was selected.

__HAL_RCC_RTC_DISABLE

RTC Clock Source

RCC_RTCCLKSOURCE_NO_CLK

No clock used as RTC clock

RCC_RTCCLKSOURCE_LSE

LSE oscillator clock used as RTC clock

RCC_RTCCLKSOURCE_LSI

LSI oscillator clock used as RTC clock

RCC_RTCCLKSOURCE_HSE_DIV32

HSE oscillator clock divided by 32 used as RTC clock

SHSI Config

RCC_SHSI_OFF

SHSI clock deactivation

RCC_SHSI_ON

SHSI clock activation

RCC Stop KernelWakeUpClock

RCC_STOP_KERWAKEUPCLOCK_MSI

MSI kernel clock selection after wake-up from STOP

RCC_STOP_KERWAKEUPCLOCK_HSI

HSI kernel clock selection after wake-up from STOP

Wake-Up from STOP Clock

RCC_STOP_WAKEUPCLOCK_MSI

MSI selection after wake-up from STOP

RCC_STOP_WAKEUPCLOCK_HSI

HSI selection after wake-up from STOP

System Clock Source

RCC_SYSCLKSOURCE_MSI

MSI selection as system clock

RCC_SYSCLKSOURCE_HSI

HSI selection as system clock

RCC_SYSCLKSOURCE_HSE

HSE selection as system clock

RCC_SYSCLKSOURCE_PLLCLK

PLL1 selection as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_MSI

MSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSI

HSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSE

HSE used as system clock

RCC_SYSCLKSOURCE_STATUS_PLLCLK

PLL1 used as system clock

System Clock Type

RCC_CLOCKTYPE_SYSCLK

SYSCLK to configure

RCC_CLOCKTYPE_HCLK

HCLK to configure

RCC_CLOCKTYPE_PCLK1

PCLK1 to configure

RCC_CLOCKTYPE_PCLK2

PCLK2 to configure

RCC_CLOCKTYPE_PCLK3

PCLK3 to configure

Timeout Values

RCC_DBP_TIMEOUT_VALUE

RCC_LSE_TIMEOUT_VALUE

RCC timicsel items

RCC_TIMIC_HSI_256

HSI/256 selected for Timer16/17 and LPTimer2

RCC_TIMIC_MSI_1024

MSIS/1024 selected for Timer16/17 and LPTimer2

RCC_TIMIC_MSI_4

MSIS/4 selected for Timer16/17 and LPTimer2

RCC_TIMIC_MSIK_4

MSIK/1024 selected for Timer16/17 and LPTimer2

RCC_TIMIC_MSIK_1024

MSIK/4 selected for Timer16/17 and LPTimer2

65 HAL RCC Extension Driver

65.1 RCCEx Firmware driver registers structures

65.1.1 RCC_PLL2InitTypeDef

RCC_PLL2InitTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL2Source***
- ***uint32_t PLL2M***
- ***uint32_t PLL2N***
- ***uint32_t PLL2P***
- ***uint32_t PLL2Q***
- ***uint32_t PLL2R***
- ***uint32_t PLL2RGE***
- ***uint32_t PLL2FRACN***
- ***uint32_t PLL2ClockOut***

Field Documentation

- ***uint32_t RCC_PLL2InitTypeDef::PLL2Source***
  RCC_PLL2Source: PLL2 entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2M***
  PLL2M: Division factor for PLL2 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLL2InitTypeDef::PLL2N***
  PLL2N: Multiplication factor for PLL2 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512
- ***uint32_t RCC_PLL2InitTypeDef::PLL2P***
  PLL2P: Division factor for system clock. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2Q***
  PLL2Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2R***
  PLL2R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL2InitTypeDef::PLL2RGE***
  PLL2RGE: PLL2 clock Input range This parameter must be a value of ***RCC_PLL_VCI_Range***
- ***uint32_t RCC_PLL2InitTypeDef::PLL2FRACN***
  PLL2FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL2 VCO It should be a value between 0 and 8191
- ***uint32_t RCC_PLL2InitTypeDef::PLL2ClockOut***
  PLL2ClockOut: specifies PLL2 output clock to be enabled. This parameter must be a value of ***RCC_PLL2_Clock_Output***

65.1.2 RCC_PLL3InitTypeDef

RCC_PLL3InitTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL3Source***
- ***uint32_t PLL3M***
- ***uint32_t PLL3N***
- ***uint32_t PLL3P***
- ***uint32_t PLL3Q***

- ***uint32_t PLL3R***
- ***uint32_t PLL3RGE***
- ***uint32_t PLL3FRACN***
- ***uint32_t PLL3ClockOut***

Field Documentation

- ***uint32_t RCC_PLL3InitTypeDef::PLL3Source***
  RCC_PLL3Source: PLL3 entry clock source. This parameter must be a value of ***RCC_PLL_Clock_Source***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3M***
  PLL3M: Division factor for PLL3 VCO input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 63
- ***uint32_t RCC_PLL3InitTypeDef::PLL3N***
  PLL3N: Multiplication factor for PLL3 VCO output clock. This parameter must be a number between Min_Data = 4 and Max_Data = 512
- ***uint32_t RCC_PLL3InitTypeDef::PLL3P***
  PLL3P: Division factor for system clock. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3Q***
  PLL3Q: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3R***
  PLL3R: Division factor for peripheral clocks. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t RCC_PLL3InitTypeDef::PLL3RGE***
  PLL3RGE: PLL3 clock Input range This parameter must be a value of ***RCC_PLL_VCI_Range***
- ***uint32_t RCC_PLL3InitTypeDef::PLL3FRACN***
  PLL3FRACN: Specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191
- ***uint32_t RCC_PLL3InitTypeDef::PLL3ClockOut***
  PLL3ClockOut: specifies PLL3 output clock to be enabled. This parameter must be a value of ***RCC_PLL3_Clock_Output***

65.1.3 PLL1_ClocksTypeDef

PLL1_ClocksTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL1_P_Frequency***
- ***uint32_t PLL1_Q_Frequency***
- ***uint32_t PLL1_R_Frequency***

Field Documentation

- ***uint32_t PLL1_ClocksTypeDef::PLL1_P_Frequency***
- ***uint32_t PLL1_ClocksTypeDef::PLL1_Q_Frequency***
- ***uint32_t PLL1_ClocksTypeDef::PLL1_R_Frequency***

65.1.4 PLL2_ClocksTypeDef

PLL2_ClocksTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL2_P_Frequency***
- ***uint32_t PLL2_Q_Frequency***
- ***uint32_t PLL2_R_Frequency***

Field Documentation

- ***uint32_t PLL2_ClocksTypeDef::PLL2_P_Frequency***
- ***uint32_t PLL2_ClocksTypeDef::PLL2_Q_Frequency***
- ***uint32_t PLL2_ClocksTypeDef::PLL2_R_Frequency***

65.1.5 PLL3_ClocksTypeDef

PLL3_ClocksTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PLL3_P_Frequency***
- ***uint32_t PLL3_Q_Frequency***
- ***uint32_t PLL3_R_Frequency***

Field Documentation

- ***uint32_t PLL3_ClocksTypeDef::PLL3_P_Frequency***
- ***uint32_t PLL3_ClocksTypeDef::PLL3_Q_Frequency***
- ***uint32_t PLL3_ClocksTypeDef::PLL3_R_Frequency***

65.1.6 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint64_t PeriphClockSelection***
- ***RCC_PLL2InitTypeDef PLL2***
- ***RCC_PLL3InitTypeDef PLL3***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t Usart2ClockSelection***
- ***uint32_t Usart3ClockSelection***
- ***uint32_t Uart4ClockSelection***
- ***uint32_t Uart5ClockSelection***
- ***uint32_t Usart6ClockSelection***
- ***uint32_t Lpuart1ClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2c2ClockSelection***
- ***uint32_t I2c3ClockSelection***
- ***uint32_t I2c4ClockSelection***
- ***uint32_t I2c5ClockSelection***
- ***uint32_t I2c6ClockSelection***
- ***uint32_t Lptim1ClockSelection***
- ***uint32_t Lptim2ClockSelection***
- ***uint32_t Lptim34ClockSelection***
- ***uint32_t Fdcan1ClockSelection***
- ***uint32_t Mdf1ClockSelection***
- ***uint32_t Adf1ClockSelection***
- ***uint32_t Sai1ClockSelection***
- ***uint32_t Sai2ClockSelection***
- ***uint32_t RngClockSelection***
- ***uint32_t SaesClockSelection***
- ***uint32_t IclkClockSelection***
- ***uint32_t SdmmcClockSelection***
- ***uint32_t AdcDacClockSelection***
- ***uint32_t Dac1ClockSelection***
- ***uint32_t OspiClockSelection***
- ***uint32_t HspiClockSelection***
- ***uint32_t Spi1ClockSelection***
- ***uint32_t Spi2ClockSelection***
- ***uint32_t Spi3ClockSelection***
- ***uint32_t RTCClockSelection***

- ***uint32_t LtdcClockSelection***
- ***uint32_t DsiClockSelection***
- ***uint32_t UsbPhyClockSelection***

Field Documentation

- ***uint64_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***RCC_PLL2InitTypeDef RCC_PeriphCLKInitTypeDef::PLL2***
  PLL2structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals
- ***RCC_PLL3InitTypeDef RCC_PeriphCLKInitTypeDef::PLL3***
  PLL3 structure parameters. This parameter will be used only when PLL2 is selected as kernel clock Source for some peripherals
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  Specifies USART1 clock source. This parameter can be a value of ***RCCEx_USART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart2ClockSelection***
  Specifies USART2 clock source. This parameter can be a value of ***RCCEx_USART2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart3ClockSelection***
  Specifies USART3 clock source. This parameter can be a value of ***RCCEx_USART3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart4ClockSelection***
  Specifies UART4 clock source. This parameter can be a value of ***RCCEx_UART4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Uart5ClockSelection***
  Specifies UART5 clock source. This parameter can be a value of ***RCCEx_UART5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart6ClockSelection***
  Specifies USART6 clock source. This parameter can be a value of ***RCCEx_USART6_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lpuart1ClockSelection***
  Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c1ClockSelection***
  Specifies I2C1 clock source. This parameter can be a value of ***RCCEx_I2C1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c2ClockSelection***
  Specifies I2C2 clock source. This parameter can be a value of ***RCCEx_I2C2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c3ClockSelection***
  Specifies I2C3 clock source. This parameter can be a value of ***RCCEx_I2C3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c4ClockSelection***
  Specifies I2C4 clock source. This parameter can be a value of ***RCCEx_I2C4_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c5ClockSelection***
  Specifies I2C5 clock source. This parameter can be a value of ***RCCEx_I2C5_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::I2c6ClockSelection***
  Specifies I2C6 clock source. This parameter can be a value of ***RCCEx_I2C6_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim1ClockSelection***
  Specifies LPTIM1 clock source. This parameter can be a value of ***RCCEx_LPTIM1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim2ClockSelection***
  Specifies LPTIM2 clock source. This parameter can be a value of ***RCCEx_LPTIM2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Lptim34ClockSelection***
  Specifies LPTIM3 and LPTIM4 clock source. This parameter can be a value of ***RCCEx_LPTIM34_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Fdcan1ClockSelection***
  Specifies FDCAN1 kernel clock source. This parameter can be a value of ***RCCEx_FDCAN1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Mdf1ClockSelection***
  Specifies MDF1 kernel clock source. This parameter can be a value of ***RCCEx_MDF1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Adf1ClockSelection***
  Specifies Adf1 kernel clock source. This parameter can be a value of ***RCCEx_ADF1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai1ClockSelection***
  Specifies SAI1 clock source. This parameter can be a value of ***RCCEx_SAI1_Clock_Source***

- ***uint32_t RCC_PeriphCLKInitTypeDef::Sai2ClockSelection***
  Specifies SAI2 clock source. This parameter can be a value of ***RCCEx_SAI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RngClockSelection***
  Specifies RNG clock source This parameter can be a value of ***RCCEx_RNG_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::SaesClockSelection***
  Specifies SAES clock source This parameter can be a value of ***RCCEx_SAES_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::IclkClockSelection***
  Specifies intermediate clock source used by USB, RNG and SDMMC1 This parameter can be a value of ***RCCEx_ICLK_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::SdmmcClockSelection***
  Specifies SDMMC1/2 clock source. This parameter can be a value of ***RCCEx_SDMMC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcDacClockSelection***
  Specifies ADC1, ADC4 and DAC interface clock source. This parameter can be a value of ***RCCEx_ADCDAC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Dac1ClockSelection***
  Specifies DAC1 interface clock source. This parameter can be a value of ***RCCEx_DAC1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::OspiClockSelection***
  Specifies OctoSPI clock source. This parameter can be a value of ***RCCEx_OSPI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::HspiClockSelection***
  Specifies HexaSPI clock source. This parameter can be a value of ***RCCEx_HSPI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi1ClockSelection***
  Specifies SPI1 clock source This parameter can be a value of ***RCCEx_SPI1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi2ClockSelection***
  Specifies SPI2 clock source This parameter can be a value of ***RCCEx_SPI2_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Spi3ClockSelection***
  Specifies SPI3 clock source This parameter can be a value of ***RCCEx_SPI3_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::LtdcClockSelection***
  Specifies LTDC clock source. This parameter can be a value of ***RCCEx_LTDC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::DsiClockSelection***
  Specifies DSI clock source. This parameter can be a value of ***RCCEx_DSI_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::UsbPhyClockSelection***
  Specifies USB PHY clock source. This parameter can be a value of ***RCCEx_USBPHY_Clock_Source***

65.1.7 RCC_CRSInitTypeDef

RCC_CRSInitTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Source***
- ***uint32_t Polarity***
- ***uint32_t ReloadValue***
- ***uint32_t ErrorLimitValue***
- ***uint32_t HSI48CalibrationValue***

Field Documentation

- ***uint32_t RCC_CRSInitTypeDef::Prescaler***
  Specifies the division factor of the SYNC signal. This parameter can be a value of ***RCCEx_CRS_SynchroDivider***
- ***uint32_t RCC_CRSInitTypeDef::Source***
  Specifies the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroSource***
- ***uint32_t RCC_CRSInitTypeDef::Polarity***
  Specifies the input polarity for the SYNC signal source. This parameter can be a value of ***RCCEx_CRS_SynchroPolarity***

- ***uint32_t RCC_CRSInitTypeDef::ReloadValue***
  Specifies the value to be loaded in the frequency error counter with each SYNC event. It can be calculated in using macro **__HAL_RCC_CRS_RELOADVALUE_CALCULATE(__FTARGET__, __FSYNC__)** This parameter must be a number between 0 and 0xFFFF or a value of ***RCCEx_CRS_ReloadValueDefault*** .
- ***uint32_t RCC_CRSInitTypeDef::ErrorLimitValue***
  Specifies the value to be used to evaluate the captured frequency error value. This parameter must be a number between 0 and 0xFF or a value of ***RCCEx_CRS_ErrorLimitDefault***
- ***uint32_t RCC_CRSInitTypeDef::HSI48CalibrationValue***
  Specifies a user-programmable trimming value to the HSI48 oscillator. This parameter must be a number between 0 and 0x7F or a value of ***RCCEx_CRS_HSI48CalibrationDefault***

65.1.8 RCC_CRSSynchroInfoTypeDef

RCC_CRSSynchroInfoTypeDef is defined in the stm32u5xx_hal_rcc_ex.h

Data Fields

- ***uint32_t ReloadValue***
- ***uint32_t HSI48CalibrationValue***
- ***uint32_t FreqErrorCapture***
- ***uint32_t FreqErrorDirection***

Field Documentation

- ***uint32_t RCC_CRSSynchroInfoTypeDef::ReloadValue***
  Specifies the value loaded in the Counter reload value. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::HSI48CalibrationValue***
  Specifies value loaded in HSI48 oscillator smooth trimming. This parameter must be a number between 0 and 0x3F
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorCapture***
  Specifies the value loaded in the .FECAP, the frequency error counter value latched in the time of the last SYNC event. This parameter must be a number between 0 and 0xFFFF
- ***uint32_t RCC_CRSSynchroInfoTypeDef::FreqErrorDirection***
  Specifies the value loaded in the .FEDIR, the counting direction of the frequency error counter latched in the time of the last SYNC event. It shows whether the actual frequency is below or above the target. This parameter must be a value of ***RCCEx_CRS_FreqErrorDirection***

65.2 RCCEx Firmware driver API description

The following section lists the various functions of the RCCEx library.

65.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

Note: *Important note: Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) are set to their reset values.*

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPLL1ClockFreq()***
- ***HAL_RCCEx_GetPLL2ClockFreq()***
- ***HAL_RCCEx_GetPLL3ClockFreq()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***

65.2.2 Extended clock management functions

This subsection provides a set of functions allowing to control the activation or deactivation of MSI PLL-mode, PLL2, PLL3, LSE CSS, Low speed clock output and clock after wake-up from STOP mode.

This section contains the following APIs:

- *HAL_RCCEx_EnablePLL2()*
- *HAL_RCCEx_DisablePLL2()*
- *HAL_RCCEx_EnablePLL3()*
- *HAL_RCCEx_DisablePLL3()*
- *HAL_RCCEx_EnableMSIPLLModeSelection()*
- *HAL_RCCEx_EnableMSIPLLFastStartup()*
- *HAL_RCCEx_DisableMSIPLLFastStartup()*
- *HAL_RCCEx_WakeUpStopCLKConfig()*
- *HAL_RCCEx_KerWakeUpStopCLKConfig()*
- *HAL_RCCEx_StandbyMSIRangeConfig()*
- *HAL_RCCEx_StandbyMSIKRangeConfig()*
- *HAL_RCCEx_EnableLSECSS()*
- *HAL_RCCEx_DisableLSECSS()*
- *HAL_RCCEx_EnableLSECSS_IT()*
- *HAL_RCCEx_LSECSS_IRQHandler()*
- *HAL_RCCEx_LSECSS_Callback()*
- *HAL_RCCEx_EnableMSIPLLUNLCK_IT()*
- *HAL_RCCEx_MSIPLLUNLCK_IRQHandler()*
- *HAL_RCCEx_MSIPLLUNLCK_Callback()*
- *HAL_RCCEx_EnableLSCO()*
- *HAL_RCCEx_DisableLSCO()*
- *HAL_RCCEx_EnableMSIPLLMode()*
- *HAL_RCCEx_DisableMSIPLLMode()*

65.2.3 Extended Clock Recovery System Control functions

For devices with Clock Recovery System feature (CRS), RCC Extension HAL driver can be used as follows:

1. In System clock config, HSI48 needs to be enabled
2. Enable CRS clock in IP MSP init which will use CRS functions
3. Call CRS functions as follows:
   a. Prepare synchronization configuration necessary for HSI48 calibration
      - Default values can be set for frequency Error Measurement (reload and error limit) and also HSI48 oscillator smooth trimming.
      - Macro __HAL_RCC_CRS_RELOADVALUE_CALCULATE can be also used to calculate directly reload value with target and synchronization frequencies values
   b. Call function HAL_RCCEx_CRSConfig which
      - Resets CRS registers to their default values.
      - Configures CRS registers with synchronization configuration
      - Enables automatic calibration and frequency error counter feature Note: When using USB LPM (Link Power Management) and the device is in Sleep mode, the periodic USB SOF will not be generated by the host. No SYNC signal will therefore be provided to the CRS to calibrate the HSI48 on the run. To guarantee the required clock precision after waking up from Sleep mode, the LSE or reference clock on the GPIOs should be used as SYNC signal.
   c. A polling function is provided to wait for complete synchronization
      - Call function HAL_RCCEx_CRSWaitSynchronization()
      - According to CRS status, user can decide to adjust again the calibration or continue application if synchronization is OK
4. User can retrieve information related to synchronization in calling function HAL_RCCEx_CRSGetSynchronizationInfo()

5. Regarding synchronization status and synchronization information, user can try a new calibration in changing synchronization configuration and call again HAL_RCCEx_CRSConfig. Note: When the SYNC event is detected during the downcounting phase (before reaching the zero value), it means that the actual frequency is lower than the target (and so, that the TRIM value should be incremented), while when it is detected during the upcounting phase it means that the actual frequency is higher (and that the TRIM value should be decremented).
6. In interrupt mode, user can resort to the available macros (__HAL_RCC_CRS_XXX_IT). Interrupts will go through CRS Handler (CRS_IRQn/CRS_IRQHandler)
   – Call function HAL_RCCEx_CRSConfig()
   – Enable CRS_IRQn (thanks to NVIC functions)
   – Enable CRS interrupt (__HAL_RCC_CRS_ENABLE_IT)
   – Implement CRS status management in the following user callbacks called from HAL_RCCEx_CRS_IRQHandler():
     ◦ HAL_RCCEx_CRS_SyncOkCallback()
     ◦ HAL_RCCEx_CRS_SyncWarnCallback()
     ◦ HAL_RCCEx_CRS_ExpectedSyncCallback()
     ◦ HAL_RCCEx_CRS_ErrorCallback()
7. To force a SYNC EVENT, user can use the function HAL_RCCEx_CRSSoftwareSynchronizationGenerate(). This function can be called before calling HAL_RCCEx_CRSConfig (for instance in Systick handler)

This section contains the following APIs:
- *HAL_RCCEx_CRSConfig()*
- *HAL_RCCEx_CRSSoftwareSynchronizationGenerate()*
- *HAL_RCCEx_CRSGetSynchronizationInfo()*
- *HAL_RCCEx_CRSWaitSynchronization()*
- *HAL_RCCEx_CRS_IRQHandler()*
- *HAL_RCCEx_CRS_SyncOkCallback()*
- *HAL_RCCEx_CRS_SyncWarnCallback()*
- *HAL_RCCEx_CRS_ExpectedSyncCallback()*
- *HAL_RCCEx_CRS_ErrorCallback()*

65.2.4 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (const RCC_PeriphCLKInitTypeDef * pPeriphClkInit)

Function description

Initialize the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **pPeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that contains a field PeriphClockSelection which can be a combination of the following values:
  – RCC_PERIPHCLK_USART1 USART1 peripheral clock
  – RCC_PERIPHCLK_USART2 USART2 peripheral clock
  – RCC_PERIPHCLK_USART3 USART3 peripheral clock
  – RCC_PERIPHCLK_UART4 UART4 peripheral clock
  – RCC_PERIPHCLK_UART5 UART5 peripheral clock
  – RCC_PERIPHCLK_USART6 USART6 peripheral clock
  – RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock
  – RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  – RCC_PERIPHCLK_I2C2 I2C2 peripheral clock
  – RCC_PERIPHCLK_I2C3 I2C3 peripheral clock
  – RCC_PERIPHCLK_I2C4 I2C4 peripheral clock
  – RCC_PERIPHCLK_I2C5 I2C5 peripheral clock
  – RCC_PERIPHCLK_I2C6 I2C6 peripheral clock
  – RCC_PERIPHCLK_LPTIM34 LPTIM34 peripheral clock
  – RCC_PERIPHCLK_LPTIM2 LPTIM2 peripheral clock
  – RCC_PERIPHCLK_SAES SAES peripheral clock (*)
  – RCC_PERIPHCLK_SAI1 SAI1 peripheral clock
  – RCC_PERIPHCLK_SAI2 SAI2 peripheral clock
  – RCC_PERIPHCLK_ADCDAC ADC1 ADC2 ADC4 DAC1 peripheral clock
  – RCC_PERIPHCLK_MDF1 MDF1 peripheral clock
  – RCC_PERIPHCLK_ADF1 ADF1 peripheral clock
  – RCC_PERIPHCLK_RTC RTC peripheral clock
  – RCC_PERIPHCLK_RNG RNG peripheral clock
  – RCC_PERIPHCLK_ICLK ICLK peripheral clock
  – RCC_PERIPHCLK_SDMMC SDMMC1 peripheral clock
  – RCC_PERIPHCLK_SPI1 SPI1 peripheral clock
  – RCC_PERIPHCLK_SPI2 SPI2 peripheral clock
  – RCC_PERIPHCLK_SPI3 SPI3 peripheral clock
  – RCC_PERIPHCLK_OSPI OSPI peripheral clock
  – RCC_PERIPHCLK_FDCAN1 FDCAN1 peripheral clock
  – RCC_PERIPHCLK_DAC1 DAC1 peripheral clock
  – RCC_PERIPHCLK_HSPI HSPI peripheral clock
  – RCC_PERIPHCLK_LTDC LTDC peripheral clock
  – RCC_PERIPHCLK_DSI DSI peripheral clock
  – RCC_PERIPHCLK_USBPHY USBPHY peripheral clock

Return values

- **HAL:** status

Notes

- Care must be taken when HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source: in this case the access to Backup domain is enabled.

HAL_RCCEx_GetPeriphCLKConfig

Function name

void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * pPeriphClkInit)

Function description

Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **pPeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(USART1, USART2, USART3, UART4, UART5, LPUART, I2C1, I2C2, I2C3, LPTIM1, LPTIM2, SAI1, SAI2, ADC1, ADC2, MDF1, MDF2, RTC, CLK48, SDMMC1, I2C4, SPI12, SPI3, OSPI, FDCAN1, DAC1).

Return values

- **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name

uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint64_t PeriphClk)

Function description

Return the peripheral clock frequency for peripherals.

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USART2 USART2 peripheral clock
  - RCC_PERIPHCLK_USART3 USART3 peripheral clock
  - RCC_PERIPHCLK_UART4 UART4 peripheral clock
  - RCC_PERIPHCLK_UART5 UART5 peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2C2 I2C2 peripheral clock
  - RCC_PERIPHCLK_I2C4 I2C4 peripheral clock
  - RCC_PERIPHCLK_SPI2 SPI2 peripheral clock
  - RCC_PERIPHCLK_LPTIM2 LPTIM2 peripheral clock
  - RCC_PERIPHCLK_SPI1 SPI1 peripheral clock
  - RCC_PERIPHCLK_FDCAN1 FDCAN1 peripheral clock
  - RCC_PERIPHCLK_ICLK ICLK peripheral clock
  - RCC_PERIPHCLK_MDF1 MDF1 peripheral clock
  - RCC_PERIPHCLK_SAI1 SAI1 peripheral clock
  - RCC_PERIPHCLK_SAI2 SAI2 peripheral clock
  - RCC_PERIPHCLK_SAES SAES peripheral clock (*)
  - RCC_PERIPHCLK_RNG RNG peripheral clock
  - RCC_PERIPHCLK_SDMMC SDMMC peripheral clock
  - RCC_PERIPHCLK_USART6 USART6 peripheral clock (*)
  - RCC_PERIPHCLK_LTDC LTDC peripheral clock (*)
  - RCC_PERIPHCLK_OSPI OSPI peripheral clock
  - RCC_PERIPHCLK_HSPI HSPI peripheral clock (*)
  - RCC_PERIPHCLK_I2C5 I2C5 peripheral clock (*)
  - RCC_PERIPHCLK_I2C6 I2C6 peripheral clock (*)
  - RCC_PERIPHCLK_USBPHY USB_OTG_HS peripheral clock (*)
  - RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock
  - RCC_PERIPHCLK_SPI3 SPI3 peripheral clock
  - RCC_PERIPHCLK_I2C3 I2C3 peripheral clock
  - RCC_PERIPHCLK_LPTIM34 LPTIM34 peripheral clock
  - RCC_PERIPHCLK_LPTIM1 LPTIM1 peripheral clock
  - RCC_PERIPHCLK_ADCDAC ADC1 ADC2 ADC4 DAC1 peripheral clock
  - RCC_PERIPHCLK_DAC1 DAC1 peripheral clock
  - RCC_PERIPHCLK_ADF1 ADF1 peripheral clock
  - RCC_PERIPHCLK_RTC RTC peripheral clock

Return values

- **Frequency:** in Hz

Notes

- Return 0 if peripheral clock identifier not managed by this API

HAL_RCCEx_GetPLL1ClockFreq

Function name

void HAL_RCCEx_GetPLL1ClockFreq (PLL1_ClocksTypeDef * PLL1_Clocks)

Function description

Returns the PLL1 clock frequencies :PLL1_P_Frequency,PLL1_R_Frequency and PLL1_Q_Frequency.

Parameters

- **PLL1_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL1 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or MSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL1CLK changes, this function must be called to update the right PLL1CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetPLL2ClockFreq

Function name

void HAL_RCCEx_GetPLL2ClockFreq (PLL2_ClocksTypeDef * PLL2_Clocks)

Function description

Returns the PLL2 clock frequencies :PLL2_P_Frequency,PLL2_R_Frequency and PLL2_Q_Frequency.

Parameters

- **PLL2_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL2 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL2CLK changes, this function must be called to update the right PLL2CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_GetPLL3ClockFreq

Function name

void HAL_RCCEx_GetPLL3ClockFreq (PLL3_ClocksTypeDef * PLL3_Clocks)

Function description

Returns the PLL3 clock frequencies :PLL3_P_Frequency,PLL3_R_Frequency and PLL3_Q_Frequency.

Parameters

- **PLL3_Clocks:** structure.

Return values

- **None:**

Notes

- The PLL3 clock frequencies computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- The function returns values based on HSE_VALUE, HSI_VALUE or CSI Value multiplied/divided by the PLL factors.
- This function can be used by the user application to compute the baud-rate for the communication peripherals or configure other parameters.
- Each time PLL3CLK changes, this function must be called to update the right PLL3CLK value. Otherwise, any configuration based on this function will be incorrect.

HAL_RCCEx_EnablePLL2

Function name

HAL_StatusTypeDef HAL_RCCEx_EnablePLL2 (const RCC_PLL2InitTypeDef * PLL2Init)

Function description

Enable PLL2.

Parameters

- **PLL2Init:** pointer to an RCC_PLL2InitTypeDef structure that contains the configuration information for the PLL2

Return values

- **HAL:** status

HAL_RCCEx_DisablePLL2

Function name

HAL_StatusTypeDef HAL_RCCEx_DisablePLL2 (void)

Function description

Disable PLL2.

Return values

- **HAL:** status

HAL_RCCEx_EnablePLL3

Function name

HAL_StatusTypeDef HAL_RCCEx_EnablePLL3 (const RCC_PLL3InitTypeDef * PLL3Init)

Function description

Enable PLL3.

Parameters

- **PLL3Init:** pointer to an RCC_PLL3InitTypeDef structure that contains the configuration information for the PLL3

Return values

- **HAL:** status

HAL_RCCEx_DisablePLL3

Function name

HAL_StatusTypeDef HAL_RCCEx_DisablePLL3 (void)

Function description

Disable PLL3.

Return values

- **HAL:** status

HAL_RCCEx_EnableMSIPLLFastStartup

Function name

HAL_StatusTypeDef HAL_RCCEx_EnableMSIPLLFastStartup (void)

Function description

Enable the fast PLL mode start-up of the MSI clock.

Return values

- **HAL:** status

Notes

- Prior to enable PLL-mode (MSIPLLEN = 1) before call HAL_RCCEx_EnableMSIPLLFastStartup.
- The fast start-up feature is not active the first time the PLL mode is selected. The fast start-up is active when the MSI in PLL mode returns from switch off.

HAL_RCCEx_DisableMSIPLLFastStartup

Function name

HAL_StatusTypeDef HAL_RCCEx_DisableMSIPLLFastStartup (void)

Function description

Disable the fast PLL mode start-up of the MSI clock.

Return values

- **HAL:** status

Notes

- the MSI fast startup mode disabled only when PLL-mode is enabled

HAL_RCCEx_EnableMSIPLLModeSelection

Function name

HAL_StatusTypeDef HAL_RCCEx_EnableMSIPLLModeSelection (uint32_t MSIPLLModeSelection)

Function description

Select which MSI output clock uses the PLL mode.

Parameters

- **MSIPLLModeSelection:** specifies which MSI output clock uses the PLL mode. This parameter can be one of the following values:
  - RCC_MSISPLL_MODE_SEL PLL mode applied to MSIS (MSI system) clock output
  - RCC_MSIKPLL_MODE_SEL PLL mode applied to MSIK (MSI kernel) clock output

Return values

- **HAL:** status

Notes

- Prior to disable PLL-mode (MSIPLLEN = 0) before call HAL_RCCEx_EnableMSIPLLModeSelection.
- The MSI kernel clock output uses the same oscillator source than the MSI system clock output, then the PLL mode is applied to the both clocks outputs.

HAL_RCCEx_WakeUpStopCLKConfig

Function name

void HAL_RCCEx_WakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator clock source for wakeup from Stop and CSS backup clock.

Parameters

- **WakeUpClk:** Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_WAKEUPCLOCK_MSI MSI oscillator selection
  - RCC_STOP_WAKEUPCLOCK_HSI HSI oscillator selection

Return values

- **None:**

Notes

- This function shall not be called after the Clock Security System on HSE has been enabled.

HAL_RCCEx_KerWakeUpStopCLKConfig

Function name

void HAL_RCCEx_KerWakeUpStopCLKConfig (uint32_t WakeUpClk)

Function description

Configure the oscillator Kernel clock source for wakeup from Stop.

Parameters

- **WakeUpClk:** Kernel Wakeup clock This parameter can be one of the following values:
  - RCC_STOP_KERWAKEUPCLOCK_MSI: MSI oscillator selection
  - RCC_STOP_KERWAKEUPCLOCK_HSI: HSI oscillator selection

Return values

- **None:**

HAL_RCCEx_StandbyMSIRangeConfig

Function name

void HAL_RCCEx_StandbyMSIRangeConfig (uint32_t MSIRange)

Function description

Configure the MSI range after standby mode.

Parameters

- **MSIRange:** MSI range This parameter can be one of the following values:
  - RCC_MSIRANGE_4 Range 4 around 4 MHz (reset value)
  - RCC_MSIRANGE_5 Range 5 around 2 MHz
  - RCC_MSIRANGE_6 Range 6 around 1.33 MHz
  - RCC_MSIRANGE_7 Range 7 around 1 MHz
  - RCC_MSIRANGE_8 Range 8 around 3.072 MHz

Return values

- **None:**

Notes

- After Standby its frequency can be selected between 3 possible values (1, 3.072 or 4 MHz).

HAL_RCCEx_StandbyMSIKRangeConfig

Function name

void HAL_RCCEx_StandbyMSIKRangeConfig (uint32_t MSIKRange)

Function description

Configure the MSIK range after standby mode.

Parameters

- **MSIKRange:** MSIK range This parameter can be one of the following values:
  - RCC_MSIKRANGE_4 Range 4 around 4 MHz (reset value)
  - RCC_MSIKRANGE_5 Range 5 around 2 MHz
  - RCC_MSIKRANGE_6 Range 6 around 1.33 MHz
  - RCC_MSIKRANGE_7 Range 7 around 1 MHz
  - RCC_MSIKRANGE_8 Range 8 around 3.072 MHz

Return values

- **None:**

Notes

- After Standby its frequency can be selected between 5 possible values (1, 1.33, 2, 3.072 or 4 MHz).

HAL_RCCEx_EnableLSECSS

Function name

void HAL_RCCEx_EnableLSECSS (void)

Function description

Enable the LSE Clock Security System.

Return values

- **None:**

Notes

- Prior to enable the LSE Clock Security System, LSE oscillator is to be enabled with HAL_RCC_OscConfig() and the LSE oscillator clock is to be selected as RTC clock with HAL_RCCEx_PeriphCLKConfig().

HAL_RCCEx_DisableLSECSS

Function name

void HAL_RCCEx_DisableLSECSS (void)

Function description

Disable the LSE Clock Security System.

Return values

- **None:**

Notes

- LSE Clock Security System can only be disabled after a LSE failure detection.

HAL_RCCEx_EnableLSECSS_IT

Function name

void HAL_RCCEx_EnableLSECSS_IT (void)

Function description

Enable the LSE Clock Security System Interrupt & corresponding EXTI line.

Return values

- **None:**

Notes

- LSE Clock Security System Interrupt is mapped on EXTI line Not available in STM32U575/585 rev. X and and STM32U59x/5Ax rev. B/Y devices.

HAL_RCCEx_EnableMSIPLLUNLCK_IT

Function name

void HAL_RCCEx_EnableMSIPLLUNLCK_IT (void)

Function description

Enable the MSI PLL Unlock Interrupt & corresponding EXTI line.

Return values

- **None:**

Notes

- MSI PLL Unlock Interrupt is mapped on EXTI line Not available in STM32U575/585 rev. X and and STM32U59x/5Ax rev. B/Y devices.

HAL_RCCEx_LSECSS_IRQHandler

Function name

void HAL_RCCEx_LSECSS_IRQHandler (void)

Function description

Handle the RCC LSE Clock Security System interrupt request.

Return values

- **None:**

Notes

- LSECSS EXTI is not available in STM32U575/585 rev. X and and STM32U59x/5Ax rev. B/Y devices.

HAL_RCCEx_LSECSS_Callback

Function name

void HAL_RCCEx_LSECSS_Callback (void)

Function description

RCCEx LSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCCEx_MSIPLLUNLCK_IRQHandler

Function name

void HAL_RCCEx_MSIPLLUNLCK_IRQHandler (void)

Function description

Handle the RCC MSI PLL Unlock interrupt request.

Return values

- **None:**

Notes

- Not available in STM32U575/585 rev. X and and STM32U59x/5Ax rev. B/Y devices.

HAL_RCCEx_MSIPLLUNLCK_Callback

Function name

void HAL_RCCEx_MSIPLLUNLCK_Callback (void)

Function description

RCCEx RCC MSI PLL Unlock interrupt callback.

Return values

- **none:**

Notes

- Not available in STM32U575/585 rev. X and and STM32U59x/5Ax rev. B/Y devices.

HAL_RCCEx_EnableLSCO

Function name

void HAL_RCCEx_EnableLSCO (uint32_t LSCOSource)

Function description

Select the Low Speed clock source to output on LSCO pin (PA2).

Parameters

- **LSCOSource:** specifies the Low Speed clock source to output. This parameter can be one of the following values:
  – RCC_LSCOSOURCE_LSI LSI clock selected as LSCO source
  – RCC_LSCOSOURCE_LSE LSE clock selected as LSCO source

Return values

- **None:**

HAL_RCCEx_DisableLSCO

Function name

void HAL_RCCEx_DisableLSCO (void)

Function description

Disable the Low Speed clock output.

Return values

- **None:**

HAL_RCCEx_EnableMSIPLLMode

Function name

void HAL_RCCEx_EnableMSIPLLMode (void)

Function description

Enable the PLL-mode of the MSI.

Return values

- **None:**

Notes

- Prior to enable the PLL-mode of the MSI for automatic hardware calibration LSE oscillator is to be enabled with HAL_RCC_OscConfig().

HAL_RCCEx_DisableMSIPLLMode

Function name

void HAL_RCCEx_DisableMSIPLLMode (void)

Function description

Disable the PLL-mode of the MSI.

Return values

- **None:**

Notes

- PLL-mode of the MSI is automatically reset when LSE oscillator is disabled.

HAL_RCCEx_CRSConfig

Function name

void HAL_RCCEx_CRSConfig (const RCC_CRSInitTypeDef *const pInit)

Function description

Start automatic synchronization for polling mode.

Parameters

- **pInit:** Pointer on RCC_CRSInitTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSSoftwareSynchronizationGenerate

Function name

void HAL_RCCEx_CRSSoftwareSynchronizationGenerate (void)

Function description

Generate the software synchronization event.

Return values

- **None:**

HAL_RCCEx_CRSGetSynchronizationInfo

Function name

void HAL_RCCEx_CRSGetSynchronizationInfo (RCC_CRSSynchroInfoTypeDef * pSynchroInfo)

Function description

Return synchronization info.

Parameters

- **pSynchroInfo:** Pointer on RCC_CRSSynchroInfoTypeDef structure

Return values

- **None:**

HAL_RCCEx_CRSWaitSynchronization

Function name

uint32_t HAL_RCCEx_CRSWaitSynchronization (uint32_t Timeout)

Function description

Wait for CRS Synchronization status.

Parameters

- **Timeout:** Duration of the timeout

Return values

- **Combination:** of Synchronization status This parameter can be a combination of the following values:
  - RCC_CRS_TIMEOUT
  - RCC_CRS_SYNCOK
  - RCC_CRS_SYNCWARN
  - RCC_CRS_SYNCERR
  - RCC_CRS_SYNCMISS
  - RCC_CRS_TRIMOVF

Notes

- Timeout is based on the maximum time to receive a SYNC event based on synchronization frequency.
- If Timeout set to HAL_MAX_DELAY, HAL_TIMEOUT will be never returned.

HAL_RCCEx_CRS_IRQHandler

Function name

void HAL_RCCEx_CRS_IRQHandler (void)

Function description

Handle the Clock Recovery System interrupt request.

Return values

- **None:**

HAL_RCCEx_CRS_SyncOkCallback

Function name

void HAL_RCCEx_CRS_SyncOkCallback (void)

Function description

RCCEx Clock Recovery System SYNCOK interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_SyncWarnCallback

Function name

void HAL_RCCEx_CRS_SyncWarnCallback (void)

Function description

RCCEx Clock Recovery System SYNCWARN interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ExpectedSyncCallback

Function name

void HAL_RCCEx_CRS_ExpectedSyncCallback (void)

Function description

RCCEx Clock Recovery System Expected SYNC interrupt callback.

Return values

- **none:**

HAL_RCCEx_CRS_ErrorCallback

Function name

void HAL_RCCEx_CRS_ErrorCallback (uint32_t Error)

Function description

RCCEx Clock Recovery System Error interrupt callback.

Parameters

- **Error:** Combination of Error status. This parameter can be a combination of the following values:
  – RCC_CRS_SYNCERR
  – RCC_CRS_SYNCMISS
  – RCC_CRS_TRIMOVF

Return values

- **none:**

65.3 RCCEx Firmware driver defines

The following section lists the various define and macros of the module.

65.3.1 RCCEx

RCCEx

ADC1 Clock Source

RCC_ADCDACCLKSOURCE_HCLK

RCC_ADCDACCLKSOURCE_SYSCLK

RCC_ADCDACCLKSOURCE_PLL2

RCC_ADCDACCLKSOURCE_HSE

RCC_ADCDACCLKSOURCE_HSI

RCC_ADCDACCLKSOURCE_MSIK

ADF1 Clock Source

RCC_ADF1CLKSOURCE_HCLK

RCC_ADF1CLKSOURCE_PLL1

RCC_ADF1CLKSOURCE_PLL3

RCC_ADF1CLKSOURCE_PIN

RCC_ADF1CLKSOURCE_MSIK

RCCEx CRS ErrorLimitDefault

RCC_CRS_ERRORLIMIT_DEFAULT

Default Frequency error limit

RCCEx CRS Extended Features

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_ENABLE

Description:

- Enable the oscillator clock for frequency error counter.

Return value:

- None

Notes:

- when the CEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_FREQ_ERROR_COUNTER_DISABLE

Description:

- Disable the oscillator clock for frequency error counter.

Return value:

- None

__HAL_RCC_CRS_AUTOMATIC_CALIB_ENABLE

Description:

- Enable the automatic hardware adjustment of TRIM bits.

Return value:

- None

Notes:

- When the AUTOTRIMEN bit is set the CRS_CFGR register becomes write-protected.

__HAL_RCC_CRS_AUTOMATIC_CALIB_DISABLE

Description:

- Enable or disable the automatic hardware adjustment of TRIM bits.

Return value:

- None

__HAL_RCC_CRS_RELOADVALUE_CALCULATE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- None

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

RCCEx CRS Flags

RCC_CRS_FLAG_SYNCOK

SYNC event OK flag

RCC_CRS_FLAG_SYNCWARN

SYNC warning flag

RCC_CRS_FLAG_ERR

Error flag

RCC_CRS_FLAG_ESYNC

Expected SYNC flag

RCC_CRS_FLAG_SYNCERR

SYNC error

RCC_CRS_FLAG_SYNCMISS

SYNC missed

RCC_CRS_FLAG_TRIMOVF

Trimming overflow or underflow

RCCEx CRS FreqErrorDirection

RCC_CRS_FREQERRORDIR_UP

Upcounting direction, the actual frequency is above the target

RCC_CRS_FREQERRORDIR_DOWN

Downcounting direction, the actual frequency is below the target

RCCEx CRS HSI48CalibrationDefault

RCC_CRS_HSI48CALIBRATION_DEFAULT

The default value is 64, which corresponds to the middle of the trimming interval. The trimming step is around 67 kHz between two consecutive TRIM steps. A higher TRIM value corresponds to a higher output frequency

RCCEx CRS Interrupt Sources

RCC_CRS_IT_SYNCOK

SYNC event OK

RCC_CRS_IT_SYNCWARN

SYNC warning

RCC_CRS_IT_ERR

Error

RCC_CRS_IT_ESYNC

Expected SYNC

RCC_CRS_IT_SYNCERR

SYNC error

RCC_CRS_IT_SYNCMISS

SYNC missed

RCC_CRS_IT_TRIMOVF

Trimming overflow or underflow

RCCEx CRS ReloadValueDefault

RCC_CRS_RELOADVALUE_DEFAULT

The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB).

RCCEx CRS Status

RCC_CRS_NONE

RCC_CRS_TIMEOUT

RCC_CRS_SYNCOK

RCC_CRS_SYNCWARN

RCC_CRS_SYNCERR

RCC_CRS_SYNCMISS

RCC_CRS_TRIMOVF

RCCEx CRS SynchroDivider

RCC_CRS_SYNC_DIV1

Synchro Signal not divided (default)

RCC_CRS_SYNC_DIV2

Synchro Signal divided by 2

RCC_CRS_SYNC_DIV4

Synchro Signal divided by 4

RCC_CRS_SYNC_DIV8

Synchro Signal divided by 8

RCC_CRS_SYNC_DIV16

Synchro Signal divided by 16

RCC_CRS_SYNC_DIV32

Synchro Signal divided by 32

RCC_CRS_SYNC_DIV64

Synchro Signal divided by 64

RCC_CRS_SYNC_DIV128

Synchro Signal divided by 128

RCCEx CRS SynchroPolarity

RCC_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

RCC_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

RCCEx CRS SynchroSource

RCC_CRS_SYNC_SOURCE_GPIO

Synchro Signal source GPIO

RCC_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

RCC_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

DAC1 Clock Source

RCC_DAC1CLKSOURCE_LSE

RCC_DAC1CLKSOURCE_LSI

DSI Clock Source

RCC_DSICLKSOURCE_DSIPHY

DSI-PHY is selected as DSI byte lane clock (usual case)

RCC_DSICLKSOURCE_PLL3

PLL3 divider P clock selected as DSI byte lane clock (low power case)

RCCEx Exported Macros

__HAL_RCC_MSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the MSI oscillator calibration value.

Parameters:

- __MSICALIBRATIONVALUE__: specifies the calibration trimming value This parameter must be a number between 0 and 0x1F.
- __MSIRANGE__: : specifies the MSI Range value This parameter can be one of the following values:
  - RCC_MSIRANGE_0
  - RCC_MSIRANGE_1
  - RCC_MSIRANGE_2
  - RCC_MSIRANGE_3
  - RCC_MSIRANGE_4
  - RCC_MSIRANGE_5
  - RCC_MSIRANGE_6
  - RCC_MSIRANGE_7
  - RCC_MSIRANGE_8
  - RCC_MSIRANGE_9
  - RCC_MSIRANGE_10
  - RCC_MSIRANGE_11
  - RCC_MSIRANGE_12
  - RCC_MSIRANGE_13
  - RCC_MSIRANGE_14
  - RCC_MSIRANGE_15

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal MSI RC.

__HAL_RCC_TIMIC_CLK_CONFIG

Description:

- Macro to configure timer input capture clock source.

Parameters:

- __TIMICSOURCE__: : specifies the TIMIC clock source This parameter can be one or a combination of the following values:
  - RCC_TIMIC_HSI_256: HSI/256 selected for Timer16/17 and LPTimer2
  - RCC_TIMIC_MSI_1024: MSIS/1024 selected for Timer16/17 and LPTimer2
  - RCC_TIMIC_MSI_4: MSIS/4 selected for Timer16/17 and LPTimer2
  - RCC_TIMIC_MSIK_1024: MSIK/1024 selected for Timer16/17 and LPTimer2
  - RCC_TIMIC_MSIK_4 : MSIK/4 selected for Timer16/17 and LPTimer2

Return value:

- None

Notes:

- combination to be avoid : RCC_TIMIC_MSI_1024 and RCC_TIMIC_MSIK_1024 RCC_TIMIC_MSI_4 and RCC_TIMIC_MSIK_4

__HAL_RCC_TIMIC_CLK_DISABLE

Description:

- Macro to disable timer input capture clock source.

Return value:

- None

__HAL_RCC_PLL2_ENABLE

Description:

- Macros to enable or disable the PLL2.

Return value:

- None

Notes:

- After enabling PLL2, the application software should wait on PLL2RDY flag to be set indicating that PLL2 clock is stable and can be used as kernel clock source. The PLL2 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL2_DISABLE

__HAL_RCC_PLL2_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL2 clock source.

Parameters:

- __PLL2SOURCE__: specifies the PLL2 entry clock source. This parameter can be one of the following values:
  – RCC_PLLSOURCE_MSI: MSI oscillator clock selected as PLL2 clock entry
  – RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL2 clock entry
  – RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL2 clock entry

Notes:

- This function must be used only when all PLL2 is disabled.

__HAL_RCC_GET_PLL2_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL2 clock source.

Return value:

- The: oscillator used as PLL2 clock source. The returned value can be one of the following:
  – RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
  – RCC_PLLSOURCE_MSI: MSI oscillator is used as PLL clock source.
  – RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  – RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_PLL2_CONFIG

Description:

- Macro to configures the PLL2 source, multiplication and division factors.

Parameters:

- __PLL2SOURCE__: specifies the PLL2 entry clock source. This parameter can be one of the following values:
  – RCC_PLLSOURCE_NONE No clock selected as PLL2 clock entry
  – RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL2 clock entry
  – RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL2 clock entry
  – RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL2 clock entry
- __PLL2M__: specifies the division factor of PLL2 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- __PLL2N__: specifies the multiplication factor for PLL2 VCO output clock This parameter must be a number between 4 and 512.
- __PLL2P__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL2Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL2R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when PLL2 is disabled.
- You have to set the PLL2N parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz. PLL2 clock frequency = f(PLL2) multiplied by PLL2N

__HAL_RCC_PLL2CLKOUT_ENABLE

Description:

- Enables or disables each clock output (PLL2_P_CLK, PLL2_Q_CLK, PLL2_R_CLK)

Parameters:

- __PLL2_CLOCKOUT__: specifies the PLL2 clock outputted. This parameter can be one or a combination of the following values:
  – RCC_PLL2_DIVP: This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
  – RCC_PLL2_DIVQ: This clock is used to generate the clock for the USB FS (48 MHz), the random number generator (<=48 MHz).
  – RCC_PLL2_DIVR: Clock used to clock ADC peripheral.

Return value:

- None

Notes:

- Enabling/disabling Those Clocks can be any time without the need to stop the PLL2, This is mainly used to save Power.

__HAL_RCC_PLL2CLKOUT_DISABLE

__HAL_RCC_GET_PLL2CLKOUT_CONFIG

Description:
- Macro to get the PLL2 clock output enable status.

Parameters:
- __PLL2_CLOCKOUT__: specifies the PLL2 clock to be outputted. This parameter can be one of the following values: This parameter can be one or a combination of the following values:
  - RCC_PLL2_DIVP: This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.
  - RCC_PLL2_DIVQ: This clock is used to generate the clock for the USB FS (48 MHz), the random number generator (<=48 MHz).
  - RCC_PLL2_DIVR: Clock used to clock ADC peripheral.

Return value:
- SET: / RESET

__HAL_RCC_PLL2FRACN_ENABLE

Description:
- Enables or disables Fractional Part Of The Multiplication Factor of PLL2 VCO.

Return value:
- None

Notes:
- Enabling/disabling Fractional Part can be any time without the need to stop the PLL2

__HAL_RCC_PLL2FRACN_DISABLE

__HAL_RCC_PLL2FRACN_CONFIG

Description:
- Macro to configures PLL2 clock Fractional Part Of The Multiplication Factor.

Parameters:
- __PLL2FRACN__: Specifies Fractional Part Of The Multiplication factor for PLL2 VCO It should be a value between 0 and 8191

Return value:
- None

Notes:
- These bits can be written at any time, allowing dynamic fine-tuning of the PLL2 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL2VCOSEL = 0 150 to 420 MHz if PLL2VCOSEL = 1

__HAL_RCC_PLL2_VCIRANGE

Description:
- Macro to select the PLL2 reference frequency range.

Parameters:
- __PLL2VCIRange__: specifies the PLL2 input frequency range This parameter can be one of the following values:
  - RCC_PLLVCIRANGE_0: Range frequency is between 4 and 8 MHz
  - RCC_PLLVCIRANGE_1: Range frequency is between 8 and 16 MHz

Return value:
- None

__HAL_RCC_PLL3_ENABLE

Notes:

- After enabling PLL3, the application software should wait on PLL3RDY flag to be set indicating that PLL3 clock is stable and can be used as kernel clock source. PLL3 is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL3_DISABLE

__HAL_RCC_PLL3_PLLSOURCE_CONFIG

Description:

- Macro to configure the PLL3 clock source.

Parameters:

- __PLL3SOURCE__: specifies the PLL3 entry clock source. This parameter can be one of the following values:
  – RCC_PLLSOURCE_MSI: MSI oscillator clock selected as PLL3 clock entry
  – RCC_PLLSOURCE_HSI: HSI oscillator clock selected as PLL3 clock entry
  – RCC_PLLSOURCE_HSE: HSE oscillator clock selected as PLL3 clock entry

Notes:

- This function must be used only when all PLL3 is disabled.

__HAL_RCC_GET_PLL3_OSCSOURCE

Description:

- Macro to get the oscillator used as PLL3 clock source.

Return value:

- The: oscillator used as PLL3 clock source. The returned value can be one of the following:
  – RCC_PLLSOURCE_NONE: No oscillator is used as PLL clock source.
  – RCC_PLLSOURCE_MSI: MSI oscillator is used as PLL clock source.
  – RCC_PLLSOURCE_HSI: HSI oscillator is used as PLL clock source.
  – RCC_PLLSOURCE_HSE: HSE oscillator is used as PLL clock source.

__HAL_RCC_PLL3_CONFIG

Description:

- Macro to configures the PLL3 source, multiplication and division factors.

Parameters:

- __PLL3SOURCE__: specifies the PLL3 entry clock source. This parameter can be one of the following values:
  – RCC_PLLSOURCE_NONE No clock selected as PLL3 clock entry
  – RCC_PLLSOURCE_MSI MSI oscillator clock selected as PLL3 clock entry
  – RCC_PLLSOURCE_HSI HSI oscillator clock selected as PLL3 clock entry
  – RCC_PLLSOURCE_HSE HSE oscillator clock selected as PLL3 clock entry
- __PLL3M__: specifies the division factor of PLL3 input clock. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- __PLL3N__: specifies the multiplication factor for PLL3 VCO output clock This parameter must be a number between 4 and 512.
- __PLL3P__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL3Q__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128
- __PLL3R__: specifies the division factor for peripheral kernel clocks This parameter must be a number between 1 and 128

Return value:

- None

Notes:

- This function must be used only when PLL3 is disabled.
- You have to set the PLL3N parameter correctly to ensure that the VCO output frequency is between 64 and 344 MHz. PLL3 clock frequency = f(PLL3) multiplied by PLL3N

__HAL_RCC_PLL3FRACN_CONFIG

Description:

- Macro to configures PLL3 clock Fractional Part of The Multiplication Factor.

Parameters:

- __PLL3FRACN__: specifies Fractional Part Of The Multiplication Factor for PLL3 VCO It should be a value between 0 and 8191

Return value:

- None

Notes:

- These bits can be written at any time, allowing dynamic fine-tuning of the PLL3 VCO
- Warning: the software has to set correctly these bits to insure that the VCO output frequency is between its valid frequency range, which is: 192 to 836 MHz if PLL3VCOSEL = 0 150 to 420 MHz if PLL3VCOSEL = 1

__HAL_RCC_PLL3_VCIRANGE

Description:

- Macro to select the PLL3 reference frequency range.

Parameters:

- __PLL3VCIRange__: specifies the PLL1 input frequency range This parameter can be one of the following values:
  – RCC_PLLVCIRANGE_0: Range frequency is between 4 and 8 MHz
  – RCC_PLLVCIRANGE_1: Range frequency is between 8 and 16 MHz

Return value:

- None

__HAL_RCC_PLL3FRACN_ENABLE

Description:
- Enables or disables Fractional Part Of The Multiplication Factor of PLL3 VCO.

Return value:
- None

Notes:
- Enabling/disabling Fractional Part can be any time without the need to stop the PLL3

__HAL_RCC_PLL3FRACN_DISABLE

__HAL_RCC_PLL3CLKOUT_ENABLE

Description:
- Enables or disables each clock output (PLL3_P_CLK, PLL3_Q_CLK, PLL3_R_CLK)

Parameters:
- __PLL3_CLOCKOUT__: specifies the PLL3 clock to be outputted This parameter can be one of the following values:
  – RCC_PLL3_DIVP: This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.

Return value:
- None

Notes:
- Enabling/disabling Those Clocks can be any time without the need to stop the PLL3, This is mainly used to save Power.

__HAL_RCC_PLL3CLKOUT_DISABLE

__HAL_RCC_GET_PLL3CLKOUT_CONFIG

Description:
- Macro to get clock output enable status (PLL3_SAI2).

Parameters:
- __PLL3_CLOCKOUT__: specifies the PLL3 clock to be outputted. This parameter can be one of the following values:
  – RCC_PLL3_DIVP: This clock is used to generate an accurate clock to achieve high-quality audio performance on SAI interface.

Return value:
- SET: / RESET

__HAL_RCC_ADCDAC_CONFIG

Description:
- Macro to configure the ADC1, ADC2, ADC4 and DAC interface clock.

Parameters:
- __ADCDAC_CLKSOURCE__: specifies the ADC1, ADC4 and DAC digital interface clock source. This parameter can be one of the following values:
  – RCC_ADCDACCLKSOURCE_HCLK clock selected as ADC1, ADC2, ADC4 and DAC clock
  – RCC_ADCDACCLKSOURCE_SYSCLK clock selected as ADC1, ADC2 ADC4 and DAC clock
  – RCC_ADCDACCLKSOURCE_PLL2 clock selected as ADC1, ADC2 ADC4 and DAC clock
  – RCC_ADCDACCLKSOURCE_HSE clock selected as ADC1, ADC2 ADC4 and DAC clock
  – RCC_ADCDACCLKSOURCE_HSI clock selected as ADC1, ADC2 ADC4 and DAC clock
  – RCC_ADCDACCLKSOURCE_MSIK clock selected as ADC1, ADC2 ADC4 and DAC clock

Return value:
- None

__HAL_RCC_GET_ADCDAC_SOURCE

Description:

- Macro to get the ADCDAC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADCDACCLKSOURCE_HCLK clock used as ADC1, ADC2, ADC4 and DAC clock
  - RCC_ADCDACCLKSOURCE_SYSCLK clock used as ADC1, ADC2, ADC4 and DAC clock
  - RCC_ADCDACCLKSOURCE_PLL2 clock used as ADC1, ADC2, ADC4 and DAC clock
  - RCC_ADCDACCLKSOURCE_HSE clock used as ADC1, ADC2, ADC4 and DAC clock
  - RCC_ADCDACCLKSOURCE_HSI clock used as ADC1, ADC2, ADC4 and DAC clock
  - RCC_ADCDACCLKSOURCE_MSIK clock used as ADC1, ADC2, ADC4 and DAC clock

__HAL_RCC_CLK48_CONFIG

Description:

- Macro to configure the CLK48 source (CLK48CLK).

Parameters:

- __CLK48_SOURCE__: specifies the CLK48 clock source. This parameter can be one of the following values:
  - RCC_CLK48CLKSOURCE_HSI48: HSI48 selected as CLK48 source
  - RCC_CLK48CLKSOURCE_PLL2 : PLL2 selected as CLK48 source
  - RCC_CLK48CLKSOURCE_PLL1 : PLL1 selected as CLK48 source
  - RCC_CLK48CLKSOURCE_MSIK : MSIK selected as CLK48 source

__HAL_RCC_GET_ICLK_SOURCE

Description:

- macro to get the CLK48 source.

Return value:

- The: clock source can be one of the following values:
  - RCC_CLK48CLKSOURCE_HSI48: HSI48 used as CLK48 source
  - RCC_CLK48CLKSOURCE_PLL2 : PLL2 used as CLK48 source
  - RCC_CLK48CLKSOURCE_PLL1 : PLL1 used as CLK48 source
  - RCC_CLK48CLKSOURCE_MSIK : MSIK used as CLK48 source

__HAL_RCC_FDCAN1_CONFIG

Description:

- Macro to configure the FDCAN1 kernel clock (FDCAN1CLK).

Parameters:

- __FDCAN1_CLKSOURCE__: specifies the FDCAN1 kernel clock source. This parameter can be one of the following values:
  - RCC_FDCAN1CLKSOURCE_HSE HSE selected as FDCAN1 kernel clock
  - RCC_FDCAN1CLKSOURCE_PLL1 PLL1 Clock selected as FDCAN1 kernel clock
  - RCC_FDCAN1CLKSOURCE_PLL2 PLL2 Clock selected as FDCAN1 kernel clock

Return value:

- None

__HAL_RCC_GET_FDCAN1_SOURCE

Description:

- Macro to get the FDCAN1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_FDCAN1CLKSOURCE_HSE HSE selected as FDCAN1 kernel clock
  - RCC_FDCAN1CLKSOURCE_PLL1 PLL1 Clock selected as FDCAN1 kernel clock
  - RCC_FDCAN1CLKSOURCE_PLL2 PLL2 Clock selected as FDCAN1 kernel clock

__HAL_RCC_LPTIM34_CONFIG

Description:

- Macro to configure the LPTIM34 clock (LPTIM34CLK).

Parameters:

- __LPTIM34_CLKSOURCE__: specifies the LPTIM34 clock source. This parameter can be one of the following values:
  - RCC_LPTIM34CLKSOURCE_MSIK MSIK selected as LPTIM34 clock
  - RCC_LPTIM34CLKSOURCE_LSI LSI selected as LPTIM34 clock
  - RCC_LPTIM34CLKSOURCE_HSI HSI selected as LPTIM34 clock
  - RCC_LPTIM34CLKSOURCE_LSE LSE selected as LPTIM34 clock

Return value:

- None

__HAL_RCC_GET_LPTIM34_SOURCE

Description:

- Macro to get the LPTIM34 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM34CLKSOURCE_MSIK MSIK selected as LPTIM34 clock
  - RCC_LPTIM34CLKSOURCE_LSI LSI selected as LPTIM34 clock
  - RCC_LPTIM34CLKSOURCE_HSI HSI selected as LPTIM34 clock
  - RCC_LPTIM34CLKSOURCE_LSE LSE selected as LPTIM34 clock

__HAL_RCC_LPTIM1_CONFIG

Description:

- Macro to configure the LPTIM1 clock (LPTIM1CLK).

Parameters:

- __LPTIM1_CLKSOURCE__: specifies the LPTIM1 clock source. This parameter can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_MSIK MSIK selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI LSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_HSI HSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock

Return value:

- None

__HAL_RCC_GET_LPTIM1_SOURCE

Description:

- Macro to get the LPTIM1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM1CLKSOURCE_MSIK MSIK selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSI HSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_HSI HSI selected as LPTIM1 clock
  - RCC_LPTIM1CLKSOURCE_LSE LSE selected as LPTIM1 clock

__HAL_RCC_LPTIM2_CONFIG

Description:

- Macro to configure the LPTIM2 clock (LPTIM2CLK).

Parameters:

- __LPTIM2_CLKSOURCE__: specifies the LPTIM2 clock source. This parameter can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK1 PCLK1 selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI LSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_HSI HSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE LSE selected as LPTIM2 clock

Return value:

- None

__HAL_RCC_GET_LPTIM2_SOURCE

Description:

- Macro to get the LPTIM2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPTIM2CLKSOURCE_PCLK1 PCLK1 selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSI HSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_HSI HSI selected as LPTIM2 clock
  - RCC_LPTIM2CLKSOURCE_LSE LSE selected as LPTIM2 clock

__HAL_RCC_SPI1_CONFIG

Description:

- macro to configure the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI1CLKSOURCE_PCLK2 : PCLK2 Clock selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_SYSCLK : SYSCLK Clock selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_HSI : HSI Clock selected as SPI1 clock
  - RCC_SPI1CLKSOURCE_MSIK : MSIK Clock selected as SPI1 clock

__HAL_RCC_GET_SPI1_SOURCE

Description:

- macro to get the SPI1 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_SPI1CLKSOURCE_PCLK2 : PCLK2 Clock used as SPI1 clock
  – RCC_SPI1CLKSOURCE_SYSCLK : SYSCLK Clock used as SPI1 clock
  – RCC_SPI1CLKSOURCE_HSI : HSI Clock used as SPI1 clock
  – RCC_SPI1CLKSOURCE_MSIK : MSIK Clock used as SPI1 clock

__HAL_RCC_SPI2_CONFIG

Description:

- macro to configure the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_SPI2CLKSOURCE_PCLK3 : PCLK3 selected as SPI2 clock
  – RCC_SPI2CLKSOURCE_SYSCLK : SYSCLK Clock selected as SPI2 clock
  – RCC_SPI2CLKSOURCE_HSI : HSI Clock selected as SPI2 clock
  – RCC_SPI2CLKSOURCE_MSIK : MSIK Clock selected as SPI2 clock

__HAL_RCC_GET_SPI2_SOURCE

Description:

- macro to get the SPI2 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_SPI2CLKSOURCE_PCLK1 : PCLK1 Clock used as SPI2 clock
  – RCC_SPI2CLKSOURCE_SYSCLK : SYSCLK Clock used as SPI2 clock
  – RCC_SPI2CLKSOURCE_HSI : HSI Clock used as SPI2 clock
  – RCC_SPI2CLKSOURCE_MSIK : MSIK Clock used as SPI2 clock

__HAL_RCC_SPI3_CONFIG

Description:

- macro to configure the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_SPI3CLKSOURCE_PCLK3 : PCLK3 selected as SPI3 clock
  – RCC_SPI3CLKSOURCE_SYSCLK : SYSCLK Clock selected as SPI3 clock
  – RCC_SPI3CLKSOURCE_HSI : HSI Clock selected as SPI3 clock
  – RCC_SPI3CLKSOURCE_MSIK : MSIK Clock selected as SPI3 clock

__HAL_RCC_GET_SPI3_SOURCE

Description:

- macro to get the SPI3 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_SPI3CLKSOURCE_PCLK3 : PCLK3 used as SPI3 clock
  – RCC_SPI3CLKSOURCE_SYSCLK : SYSCLK Clock used as SPI3 clock
  – RCC_SPI3CLKSOURCE_HSI : HSI Clock used as SPI3 clock
  – RCC_SPI3CLKSOURCE_MSIK : MSIK Clock used as SPI3 clock

__HAL_RCC_I2C1_CONFIG

Description:

- Macro to configure the I2C1 clock (I2C1CLK).

Parameters:

- __I2C1_CLKSOURCE__: specifies the I2C1 clock source. This parameter can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_MSIK MSIK selected as I2C1 clock

Return value:

- None

__HAL_RCC_GET_I2C1_SOURCE

Description:

- Macro to get the I2C1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C1CLKSOURCE_PCLK1 PCLK1 selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_HSI HSI selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_MSIK MSIK selected as I2C1 clock

__HAL_RCC_I2C2_CONFIG

Description:

- Macro to configure the I2C2 clock (I2C2CLK).

Parameters:

- __I2C2_CLKSOURCE__: specifies the I2C2 clock source. This parameter can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK System Clock selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_MSIK MSIK selected as I2C2 clock

Return value:

- None

__HAL_RCC_GET_I2C2_SOURCE

Description:

- Macro to get the I2C2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2C2CLKSOURCE_PCLK1 PCLK1 selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_SYSCLK System Clock selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_HSI HSI selected as I2C2 clock
  - RCC_I2C2CLKSOURCE_MSIK MSIK selected as I2C2 clock

__HAL_RCC_I2C3_CONFIG

Description:

- Macro to configure the I2C3 clock (I2C3CLK).

Parameters:

- __I2C3_CLKSOURCE__: specifies the I2C3 clock source. This parameter can be one of the following values:
  – RCC_I2C3CLKSOURCE_PCLK3 : PCLK3 selected as I2C3 clock
  – RCC_I2C3CLKSOURCE_HSI : HSI selected as I2C3 clock
  – RCC_I2C3CLKSOURCE_MSIK : MSIK selected as I2C3 clock
  – RCC_I2C3CLKSOURCE_SYSCLK : System Clock selected as I2C3 clock

Return value:

- None

__HAL_RCC_GET_I2C3_SOURCE

Description:

- Macro to get the I2C3 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_I2C3CLKSOURCE_PCLK3 : PCLK3 used as I2C3 clock
  – RCC_I2C3CLKSOURCE_HSI : HSI used as I2C3 clock
  – RCC_I2C3CLKSOURCE_MSIK : MSIK used as I2C3 clock
  – RCC_I2C3CLKSOURCE_SYSCLK : System Clock used as I2C3 clock

__HAL_RCC_I2C4_CONFIG

Description:

- Macro to configure the I2C4 clock (I2C4CLK).

Parameters:

- __I2C4_CLKSOURCE__: specifies the I2C4 clock source. This parameter can be one of the following values:
  – RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  – RCC_I2C4CLKSOURCE_SYSCLK System Clock selected as I2C4 clock
  – RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  – RCC_I2C4CLKSOURCE_MSIK MSIK selected as I2C4 clock

Return value:

- None

__HAL_RCC_GET_I2C4_SOURCE

Description:

- Macro to get the I2C4 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_I2C4CLKSOURCE_PCLK1 PCLK1 selected as I2C4 clock
  – RCC_I2C4CLKSOURCE_SYSCLK System Clock selected as I2C4 clock
  – RCC_I2C4CLKSOURCE_HSI HSI selected as I2C4 clock
  – RCC_I2C4CLKSOURCE_MSIK MSIK selected as I2C4 clock

__HAL_RCC_I2C5_CONFIG

Description:
- Macro to configure the I2C5 clock (I2C5CLK).

Parameters:
- __I2C5_CLKSOURCE__: specifies the I2C5 clock source. This parameter can be one of the following values:
  – RCC_I2C5CLKSOURCE_PCLK1 PCLK1 selected as I2C5 clock
  – RCC_I2C5CLKSOURCE_SYSCLK System Clock selected as I2C5 clock
  – RCC_I2C5CLKSOURCE_HSI HSI selected as I2C5 clock
  – RCC_I2C5CLKSOURCE_MSIK MSIK selected as I2C5 clock

Return value:
- None

__HAL_RCC_GET_I2C5_SOURCE

Description:
- Macro to get the I2C5 clock source.

Return value:
- The: clock source can be one of the following values:
  – RCC_I2C5CLKSOURCE_PCLK1 PCLK1 selected as I2C5 clock
  – RCC_I2C5CLKSOURCE_SYSCLK System Clock selected as I2C5 clock
  – RCC_I2C5CLKSOURCE_HSI HSI selected as I2C5 clock
  – RCC_I2C5CLKSOURCE_MSIK MSIK selected as I2C5 clock

__HAL_RCC_I2C6_CONFIG

Description:
- Macro to configure the I2C6 clock (I2C6CLK).

Parameters:
- __I2C6_CLKSOURCE__: specifies the I2C6 clock source. This parameter can be one of the following values:
  – RCC_I2C6CLKSOURCE_PCLK1 PCLK1 selected as I2C6 clock
  – RCC_I2C6CLKSOURCE_SYSCLK System Clock selected as I2C6 clock
  – RCC_I2C6CLKSOURCE_HSI HSI selected as I2C6 clock
  – RCC_I2C6CLKSOURCE_MSIK MSIK selected as I2C6 clock

Return value:
- None

__HAL_RCC_GET_I2C6_SOURCE

Description:
- Macro to get the I2C6 clock source.

Return value:
- The: clock source can be one of the following values:
  – RCC_I2C6CLKSOURCE_PCLK1 PCLK1 selected as I2C6 clock
  – RCC_I2C6CLKSOURCE_SYSCLK System Clock selected as I2C6 clock
  – RCC_I2C6CLKSOURCE_HSI HSI selected as I2C6 clock
  – RCC_I2C6CLKSOURCE_MSIK MSIK selected as I2C6 clock

__HAL_RCC_USART1_CONFIG

Description:

- Macro to configure the USART1 clock (USART1CLK).

Parameters:

- __USART1_CLKSOURCE__: specifies the USART1 clock source. This parameter can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK1 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE SE selected as USART1 clock

Return value:

- None

__HAL_RCC_GET_USART1_SOURCE

Description:

- Macro to get the USART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART1CLKSOURCE_PCLK2 PCLK1 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSI HSI selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_USART2_CONFIG

Description:

- Macro to configure the USART2 clock (USART2CLK).

Parameters:

- __USART2_CLKSOURCE__: specifies the USART2 clock source. This parameter can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

Return value:

- None

__HAL_RCC_GET_USART2_SOURCE

Description:

- Macro to get the USART2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART2CLKSOURCE_PCLK1 PCLK1 selected as USART2 clock
  - RCC_USART2CLKSOURCE_HSI HSI selected as USART2 clock
  - RCC_USART2CLKSOURCE_SYSCLK System Clock selected as USART2 clock
  - RCC_USART2CLKSOURCE_LSE LSE selected as USART2 clock

__HAL_RCC_USART3_CONFIG

Description:

- Macro to configure the USART3 clock (USART3CLK).

Parameters:

- __USART3_CLKSOURCE__: specifies the USART3 clock source. This parameter can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

Return value:

- None

__HAL_RCC_GET_USART3_SOURCE

Description:

- Macro to get the USART3 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART3CLKSOURCE_PCLK1 PCLK1 selected as USART3 clock
  - RCC_USART3CLKSOURCE_HSI HSI selected as USART3 clock
  - RCC_USART3CLKSOURCE_SYSCLK System Clock selected as USART3 clock
  - RCC_USART3CLKSOURCE_LSE LSE selected as USART3 clock

__HAL_RCC_UART4_CONFIG

Description:

- Macro to configure the UART4 clock (UART4CLK).

Parameters:

- __UART4_CLKSOURCE__: specifies the UART4 clock source. This parameter can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK System Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

Return value:

- None

__HAL_RCC_GET_UART4_SOURCE

Description:

- Macro to get the UART4 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART4CLKSOURCE_PCLK1 PCLK1 selected as UART4 clock
  - RCC_UART4CLKSOURCE_HSI HSI selected as UART4 clock
  - RCC_UART4CLKSOURCE_SYSCLK System Clock selected as UART4 clock
  - RCC_UART4CLKSOURCE_LSE LSE selected as UART4 clock

__HAL_RCC_UART5_CONFIG

Description:

- Macro to configure the UART5 clock (UART5CLK).

Parameters:

- __UART5_CLKSOURCE__: specifies the UART5 clock source. This parameter can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

Return value:

- None

__HAL_RCC_GET_UART5_SOURCE

Description:

- Macro to get the UART5 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_UART5CLKSOURCE_PCLK1 PCLK1 selected as UART5 clock
  - RCC_UART5CLKSOURCE_HSI HSI selected as UART5 clock
  - RCC_UART5CLKSOURCE_SYSCLK System Clock selected as UART5 clock
  - RCC_UART5CLKSOURCE_LSE LSE selected as UART5 clock

__HAL_RCC_USART6_CONFIG

Description:

- Macro to configure the USART6 clock (USART6CLK).

Parameters:

- __USART6_CLKSOURCE__: specifies the USART6 clock source. This parameter can be one of the following values:
  - RCC_USART6CLKSOURCE_PCLK1 PCLK1 selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI HSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_SYSCLK System Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE LSE selected as USART6 clock

Return value:

- None

__HAL_RCC_GET_USART6_SOURCE

Description:

- Macro to get the USART6 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USART6CLKSOURCE_PCLK1 PCLK1 selected as USART6 clock
  - RCC_USART6CLKSOURCE_HSI HSI selected as USART6 clock
  - RCC_USART6CLKSOURCE_SYSCLK System Clock selected as USART6 clock
  - RCC_USART6CLKSOURCE_LSE LSE selected as USART6 clock

__HAL_RCC_LPUART1_CONFIG

Description:
- Macro to configure the LPUART1 clock (LPUART1CLK).

Parameters:
- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  – RCC_LPUART1CLKSOURCE_PCLK3 PCLK3 selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_MSIK MSIK selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

Return value:
- None

__HAL_RCC_GET_LPUART1_SOURCE

Description:
- Macro to get the LPUART1 clock source.

Return value:
- The: clock source can be one of the following values:
  – RCC_LPUART1CLKSOURCE_PCLK3 PCLK3 selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_HSI HSI selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_MSIK MSIK selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_SYSCLK System Clock selected as LPUART1 clock
  – RCC_LPUART1CLKSOURCE_LSE LSE selected as LPUART1 clock

__HAL_RCC_OSPI_CONFIG

Description:
- Macro to configure the OctoSPI clock.

Parameters:
- __OSPI_CLKSOURCE__: specifies the OctoSPI clock source. This parameter can be one of the following values:
  – RCC_OSPICLKSOURCE_SYSCLK System Clock selected as OctoSPI clock
  – RCC_OSPICLKSOURCE_MSIK MSIK clock selected as OctoSPI clock
  – RCC_OSPICLKSOURCE_PLL1 PLL1 Q divider clock selected as OctoSPI clock
  – RCC_OSPICLKSOURCE_PLL2 PLL2 Q divider clock selected as OctoSPI clock

Return value:
- None

__HAL_RCC_GET_OSPI_SOURCE

Description:
- Macro to get the OctoSPI clock source.

Return value:
- The: clock source can be one of the following values:
  – RCC_OSPICLKSOURCE_SYSCLK System Clock selected as OctoSPI clock
  – RCC_OSPICLKSOURCE_MSIK MSIK clock selected as OctoSPI clock
  – RCC_OSPICLKSOURCE_PLL1 PLL1 Q divider clock selected as OctoSPI clock
  – RCC_OSPICLKSOURCE_PLL2 PLL2 Q divider clock selected as OctoSPI clock

__HAL_RCC_HSPI_CONFIG

Description:

- Macro to configure the HexaSPI clock.

Parameters:

- __HSPI_CLKSOURCE__: specifies the HexaSPI clock source. This parameter can be one of the following values:
  – RCC_HSPICLKSOURCE_SYSCLK System Clock selected as HexaSPI clock
  – RCC_HSPICLKSOURCE_PLL1 PLL1 Q divider clock selected as HexaSPI clock
  – RCC_HSPICLKSOURCE_PLL2 PLL2 Q divider clock selected as HexaSPI clock
  – RCC_HSPICLKSOURCE_PLL3 PLL3 R divider clock selected as HexaSPI clock

Return value:

- None

__HAL_RCC_GET_HSPI_SOURCE

Description:

- Macro to get the HexaSPI clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_HSPICLKSOURCE_SYSCLK System Clock selected as HexaSPI clock
  – RCC_HSPICLKSOURCE_PLL1 PLL1 Q divider clock selected as HexaSPI clock
  – RCC_HSPICLKSOURCE_PLL2 PLL2 Q divider clock selected as HexaSPI clock
  – RCC_HSPICLKSOURCE_PLL3 PLL3 R divider clock selected as HexaSPI clock

__HAL_RCC_SDMMC_CONFIG

Description:

- Macro to configure the SDMMC1/2 clock (SDMMCCLK).

Parameters:

- __SDMMC_CLKSOURCE__: specifies the SDMMC1/2 clock source. This parameter can be one of the following values:
  – RCC_SDMMCCLKSOURCE_CLK48: CLK48 selected as SDMMC1/2 clock
  – RCC_SDMMCCLKSOURCE_PLL1: PLL1 P selected as SDMMC1/2 clock

__HAL_RCC_GET_SDMMC_SOURCE

Description:

- macro to get the SDMMC1/2 clock source.

Return value:

- The: clock source can be one of the following values:
  – RCC_SDMMCCLKSOURCE_CLK48: CLK48 selected as SDMMC1/2 clock
  – RCC_SDMMCCLKSOURCE_PLL1: PLL1 P selected as SDMMC1/2 clock

__HAL_RCC_RNG_CONFIG

Description:

- macro to configure the RNG clock (RNGCLK).

Parameters:

- __RNG_CLKSource__: specifies the RNG clock source. This parameter can be one of the following values:
  – RCC_RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock
  – RCC_RNGCLKSOURCE_HSI48_DIV2: HSI48/2 selected as RNG clock
  – RCC_RNGCLKSOURCE_HSI: HSI selected as RNG clock

__HAL_RCC_GET_RNG_SOURCE

Description:
- macro to get the RNG clock source.

Return value:
- The: clock source can be one of the following values:
  – RCC_RNGCLKSOURCE_HSI48: HSI48 selected as RNG clock
  – RCC_RNGCLKSOURCE_HSI48_DIV2: HSI48/2 selected as RNG clock
  – RCC_RNGCLKSOURCE_HSI: HSI selected as RNG clock

__HAL_RCC_SAES_CONFIG

Description:
- macro to configure the SAES clock (SAESCLK).

Parameters:
- __SAES_CLKSource__: specifies the SAES clock source. This parameter can be one of the following values:
  – RCC_SAESCLKSOURCE_SHSI: SHSI selected as SAES clock
  – RCC_SAESCLKSOURCE_SHSI_DIV2: SHSI/2 selected as SAES clock

__HAL_RCC_GET_SAES_SOURCE

Description:
- macro to get the SAES clock source.

Return value:
- The: clock source can be one of the following values:
  – RCC_SAESCLKSOURCE_SHSI: SHSI selected as SAES clock
  – RCC_SAESCLKSOURCE_SHSI_DIV2: SHSI/2 selected as SAES clock

__HAL_RCC_SAI1_CONFIG

Description:
- Macro to configure the SAI1 clock source.

Parameters:
- __SAI1_CLKSOURCE__: defines the SAI1 clock source. This clock is derived from the PLL2, system PLL or external clock (through a dedicated pin). This parameter can be one of the following values:
  – RCC_SAI1CLKSOURCE_PLL2 SAI1 clock = PLL2 "P" clock (PLL2CLK)
  – RCC_SAI1CLKSOURCE_PLL3 SAI1 clock = PLL3 "P" clock (PLL3CLK)
  – RCC_SAI1CLKSOURCE_PLL1 SAI1 clock = PLL1 "P" clock (PLL1CLK)
  – RCC_SAI1CLKSOURCE_PIN SAI1 clock = External Clock (SAI1_EXTCLK)
  – RCC_SAI1CLKSOURCE_HSI SAI1 clock = HSI16

Return value:
- None

Notes:
- HSI16 is automatically set as SAI1 clock source when PLL are disabled for devices without PLL3.

__HAL_RCC_GET_SAI1_SOURCE

Description:

- Macro to get the SAI1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI1CLKSOURCE_PLL2 SAI1 clock = PLL2 "P" clock (PLL2CLK)
  - RCC_SAI1CLKSOURCE_PLL3 SAI1 clock = PLL3 "P" clock (PLL3CLK)
  - RCC_SAI1CLKSOURCE_PLL1 SAI1 clock = PLL "P" clock (PLL1CLK)
  - RCC_SAI1CLKSOURCE_PIN SAI1 clock = External Clock (SAI1_EXTCLK)
  - RCC_SAI1CLKSOURCE_HSI SAI1 clock = HSI16

Notes:

- Despite returned values RCC_SAI1CLKSOURCE_PLL2 or RCC_SAI1CLKSOURCE_PLL, HSI16 is automatically set as SAI1 clock source when PLLs are disabled for devices without PLL3.

__HAL_RCC_SAI2_CONFIG

Description:

- Macro to configure the SAI2 clock source.

Parameters:

- __SAI2_CLKSOURCE__: defines the SAI2 clock source. This clock is derived from the PLL3, system PLL or external clock (through a dedicated pin). This parameter can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLL2 SAI2 clock = PLL2 "P" clock (PLL2CLK)
  - RCC_SAI2CLKSOURCE_PLL3 SAI2 clock = PLL3 "P" clock (PLL3CLK)
  - RCC_SAI2CLKSOURCE_PLL1 SAI2 clock = PLL1 "P" clock (PLL1CLK)
  - RCC_SAI2CLKSOURCE_PIN SAI2 clock = External Clock (SAI2_EXTCLK)
  - RCC_SAI2CLKSOURCE_HSI SAI2 clock = HSI16

Return value:

- None

__HAL_RCC_GET_SAI2_SOURCE

Description:

- Macro to get the SAI2 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SAI2CLKSOURCE_PLL2 SAI2 clock = PLL2 "P" clock (PLL2CLK)
  - RCC_SAI2CLKSOURCE_PLL3 SAI2 clock = PLL3 "P" clock (PLL3CLK)
  - RCC_SAI2CLKSOURCE_PLL1 SAI2 clock = PLL1 "P" clock (PLL1CLK)
  - RCC_SAI2CLKSOURCE_PIN SAI2 clock = External Clock (SAI2_EXTCLK)
  - RCC_SAI2CLKSOURCE_HSI SAI2 clock = HSI16

__HAL_RCC_MDF1_CONFIG

Description:

- Macro to configure the MDF1 clock.

Parameters:

- __MDF1_CLKSOURCE__: specifies the MDF1 clock source. This parameter can be one of the following values:
  - RCC_MDF1CLKSOURCE_HCLK HCLK Clock selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_PLL1 PLL1 P Clock selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_PLL3 PLL3 Q Clock selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_PIN External Clock (SAI1_EXTCLK) selected as MDF1 clock
  - RCC_MDF1CLKSOURCE_MSIK MSIK Clock selected as MDF1 clock

Return value:

- None

__HAL_RCC_GET_MDF1_SOURCE

Description:

- Macro to get the MDF1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_MDF1CLKSOURCE_HCLK HCLK Clock used as MDF1 clock
  - RCC_MDF1CLKSOURCE_PLL1 PLL1 Clock used as MDF1 clock
  - RCC_MDF1CLKSOURCE_PLL3 PLL3 Clock used as MDF1 clock
  - RCC_MDF1CLKSOURCE_PIN External Clock (SAI1_EXTCLK) used as MDF1 clock
  - RCC_MDF1CLKSOURCE_MSIK MSIK Clock used as MDF1 clock

__HAL_RCC_ADF1_CONFIG

Description:

- Macro to configure the ADF1 clock.

Parameters:

- __ADF1_CLKSOURCE__: specifies the ADF1 clock source. This parameter can be one of the following values:
  - RCC_ADF1CLKSOURCE_HCLK HCLK Clock selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_PLL1 PLL1 P Clock selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_PLL3 PLL3 Q Clock selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_PIN External Clock (SAI1_EXTCLK) selected as ADF1 clock
  - RCC_ADF1CLKSOURCE_MSIK MSI Clock selected as ADF1 clock

Return value:

- None

__HAL_RCC_GET_ADF1_SOURCE

Description:

- Macro to get the ADF1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADF1CLKSOURCE_HCLK HCLK Clock used as ADF1 clock
  - RCC_ADF1CLKSOURCE_PLL1 PLL1 P Clock used as ADF1 clock
  - RCC_ADF1CLKSOURCE_PLL3 PLL3 Q Clock used as ADF1 clock
  - RCC_ADF1CLKSOURCE_PIN External Clock (SAI1_EXTCLK) used as ADF1 clock
  - RCC_ADF1CLKSOURCE_MSIK MSI Clock used as ADF1 clock

__HAL_RCC_DAC1_CONFIG

Description:

- Macro to configure the DAC1 interface clock.

Parameters:

- __DAC1_CLKSOURCE__: specifies the DAC1 digital interface clock source. This parameter can be one of the following values:
  - RCC_DAC1CLKSOURCE_LSE LSE clock selected as DAC1 clock
  - RCC_DAC1CLKSOURCE_LSI LSI clock selected as DAC1 clock

Return value:

- None

__HAL_RCC_GET_DAC1_SOURCE

Description:

- Macro to get the DAC1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_DAC1CLKSOURCE_LSE LSE clock selected as DAC1 clock
  - RCC_DAC1CLKSOURCE_LSI LSI clock selected as DAC1 clock

__HAL_RCC_LTDC_CONFIG

Description:

- Macro to configure the LTDC clock.

Parameters:

- __LTDC_CLKSOURCE__: specifies the LTDC clock source. This parameter can be one of the following values:
  - RCC_LTDCCLKSOURCE_PLL3 PLL3 divider R clock selected as LTDC kernel clock
  - RCC_LTDCCLKSOURCE_PLL2 PLL2 divider R clock selected as LTDC kernel clock

Return value:

- None

__HAL_RCC_GET_LTDC_SOURCE

Description:

- Macro to get the LTDC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LTDCCLKSOURCE_PLL3 PLL3 divider R clock selected as LTDC kernel clock
  - RCC_LTDCCLKSOURCE_PLL2 PLL2 divider R clock selected as LTDC kernel clock

__HAL_RCC_DSI_CONFIG

Description:

- Macro to configure the DSI clock.

Parameters:

- __DSI_CLKSOURCE__: specifies the DSI clock source. This parameter can be one of the following values:
  - RCC_DSICLKSOURCE_DSIPHY DSI-PHY clock selected as DSI clock
  - RCC_DSICLKSOURCE_PLL3 PLL3 divider P clock selected as DSI clock (low power case)

Return value:

- None

__HAL_RCC_GET_DSI_SOURCE

Description:

- Macro to get the DSI clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_DSICLKSOURCE_DSIPHY DSI-PHY clock selected as DSI clock
  - RCC_DSICLKSOURCE_PLL3 PLL3 divider P clock selected as DSI clock (low power case)

__HAL_RCC_USBPHY_CONFIG

Description:

- Macro to configure the USB PHY clock.

Parameters:

- __USBPHY_CLKSOURCE__: specifies the USB PHY clock source. This parameter can be one of the following values:
  - RCC_USBPHYCLKSOURCE_HSE HSE clock selected as USB PHY clock
  - RCC_USBPHYCLKSOURCE_HSE_DIV2 HSE clock div by 2 selected as USB PHY clock
  - RCC_USBPHYCLKSOURCE_PLL1 PLL1 P divider clock selected as USB PHY clock
  - RCC_USBPHYCLKSOURCE_PLL1_DIV2 PLL1 P divider clock div by 2 selected as USB PHY clock

Return value:

- None

__HAL_RCC_GET_USBPHY_SOURCE

Description:

- Macro to get the USB PHY clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_USBPHYCLKSOURCE_HSE HSE clock selected as USB PHY clock
  - RCC_USBPHYCLKSOURCE_HSE_DIV2 HSE clock div by 2 selected as USB PHY clock
  - RCC_USBPHYCLKSOURCE_PLL1 PLL1 P divider clock selected as USB PHY clock
  - RCC_USBPHYCLKSOURCE_PLL1_DIV2 PLL1 P divider clock div by 2 selected as USB PHY clock

__HAL_RCC_DPHY_CONFIG

Description:

- Macro to configure the DPHY clock.

Parameters:

- __PRESCALER__: specifies the DPHY clock source prescaler. This parameter can be one of the following values:
  - RCC_HCLK_DIV1 HCLK divided by 1 selected as DPHY clock
  - RCC_HCLK_DIV2 HCLK divided by 2 selected as DPHY clock
  - RCC_HCLK_DIV4 HCLK divided by 4 selected as DPHY clock
  - RCC_HCLK_DIV8 HCLK divided by 8 selected as DPHY clock
  - RCC_HCLK_DIV16 HCLK divided by 16 selected as DPHY clock

Return value:

- None

__HAL_RCC_GET_DPHY_CONFIG

Description:
- Macro to get the DPHY clock prescaler configuration.

Return value:
- The: clock source prescaler can be one of the following values:
  - RCC_HCLK_DIV1 HCLK divided by 1 selected as DPHY clock
  - RCC_HCLK_DIV2 HCLK divided by 2 selected as DPHY clock
  - RCC_HCLK_DIV4 HCLK divided by 4 selected as DPHY clock
  - RCC_HCLK_DIV8 HCLK divided by 8 selected as DPHY clock
  - RCC_HCLK_DIV16 HCLK divided by 16 selected as DPHY clock
- None

__HAL_RCC_CRS_ENABLE_IT

Description:
- Enable the specified CRS interrupts.

Parameters:
- __INTERRUPT__: specifies the CRS interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:
- None

__HAL_RCC_CRS_DISABLE_IT

Description:
- Disable the specified CRS interrupts.

Parameters:
- __INTERRUPT__: specifies the CRS interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:
- None

__HAL_RCC_CRS_GET_IT_SOURCE

Description:
- Check whether the CRS interrupt has occurred or not.

Parameters:
- __INTERRUPT__: specifies the CRS interrupt source to check. This parameter can be one of the following values:
  - RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  - RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  - RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  - RCC_CRS_IT_ESYNC Expected SYNC interrupt

Return value:
- The: new state of __INTERRUPT__ (SET or RESET).

RCC_CRS_IT_ERROR_MASK

Description:

- Clear the CRS interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  – RCC_CRS_IT_SYNCOK SYNC event OK interrupt
  – RCC_CRS_IT_SYNCWARN SYNC warning interrupt
  – RCC_CRS_IT_ERR Synchronization or trimming error interrupt
  – RCC_CRS_IT_ESYNC Expected SYNC interrupt
  – RCC_CRS_IT_TRIMOVF Trimming overflow or underflow interrupt
  – RCC_CRS_IT_SYNCERR SYNC error interrupt
  – RCC_CRS_IT_SYNCMISS SYNC missed interrupt

__HAL_RCC_CRS_CLEAR_IT

__HAL_RCC_CRS_GET_FLAG

Description:

- Check whether the specified CRS flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – RCC_CRS_FLAG_SYNCOK SYNC event OK
  – RCC_CRS_FLAG_SYNCWARN SYNC warning
  – RCC_CRS_FLAG_ERR Error
  – RCC_CRS_FLAG_ESYNC Expected SYNC
  – RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  – RCC_CRS_FLAG_SYNCERR SYNC error
  – RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- The: new state of _FLAG_ (TRUE or FALSE).

RCC_CRS_FLAG_ERROR_MASK

Description:

- Clear the CRS specified FLAG.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  – RCC_CRS_FLAG_SYNCOK SYNC event OK
  – RCC_CRS_FLAG_SYNCWARN SYNC warning
  – RCC_CRS_FLAG_ERR Error
  – RCC_CRS_FLAG_ESYNC Expected SYNC
  – RCC_CRS_FLAG_TRIMOVF Trimming overflow or underflow
  – RCC_CRS_FLAG_SYNCERR SYNC error
  – RCC_CRS_FLAG_SYNCMISS SYNC missed

Return value:

- None

Notes:

- RCC_CRS_FLAG_ERR clears RCC_CRS_FLAG_TRIMOVF, RCC_CRS_FLAG_SYNCERR, RCC_CRS_FLAG_SYNCMISS and consequently RCC_CRS_FLAG_ERR

__HAL_RCC_CRS_CLEAR_FLAG

FDCAN1 Kernel Clock Source

RCC_FDCAN1CLKSOURCE_HSE

RCC_FDCAN1CLKSOURCE_PLL1

RCC_FDCAN1CLKSOURCE_PLL2

HexaSPI Clock Source

RCC_HSPICLKSOURCE_SYSCLK

RCC_HSPICLKSOURCE_PLL1

RCC_HSPICLKSOURCE_PLL2

RCC_HSPICLKSOURCE_PLL3

I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

RCC_I2C1CLKSOURCE_SYSCLK

RCC_I2C1CLKSOURCE_HSI

RCC_I2C1CLKSOURCE_MSIK

I2C2 Clock Source

RCC_I2C2CLKSOURCE_PCLK1

RCC_I2C2CLKSOURCE_SYSCLK

RCC_I2C2CLKSOURCE_HSI

RCC_I2C2CLKSOURCE_MSIK

I2C3 Clock Source

RCC_I2C3CLKSOURCE_PCLK3

RCC_I2C3CLKSOURCE_SYSCLK

RCC_I2C3CLKSOURCE_HSI

RCC_I2C3CLKSOURCE_MSIK

I2C4 Clock Source

RCC_I2C4CLKSOURCE_PCLK1

RCC_I2C4CLKSOURCE_SYSCLK

RCC_I2C4CLKSOURCE_HSI

RCC_I2C4CLKSOURCE_MSIK

I2C5 Clock Source

RCC_I2C5CLKSOURCE_PCLK1

RCC_I2C5CLKSOURCE_SYSCLK

RCC_I2C5CLKSOURCE_HSI

RCC_I2C5CLKSOURCE_MSIK

I2C6 Clock Source

RCC_I2C6CLKSOURCE_PCLK1

RCC_I2C6CLKSOURCE_SYSCLK

RCC_I2C6CLKSOURCE_HSI

RCC_I2C6CLKSOURCE_MSIK

ICLK Clock Source

RCC_ICLK_CLKSOURCE_HSI48

RCC_ICLK_CLKSOURCE_PLL2

RCC_ICLK_CLKSOURCE_PLL1

RCC_ICLK_CLKSOURCE_MSIK

LPTIM1 Clock Source

RCC_LPTIM1CLKSOURCE_MSIK

RCC_LPTIM1CLKSOURCE_LSI

RCC_LPTIM1CLKSOURCE_HSI

RCC_LPTIM1CLKSOURCE_LSE

LPTIM2 Clock Source

RCC_LPTIM2CLKSOURCE_PCLK1

RCC_LPTIM2CLKSOURCE_LSI

RCC_LPTIM2CLKSOURCE_HSI

RCC_LPTIM2CLKSOURCE_LSE

LPTIM34 Clock Source

RCC_LPTIM34CLKSOURCE_MSIK

RCC_LPTIM34CLKSOURCE_LSI

RCC_LPTIM34CLKSOURCE_HSI

RCC_LPTIM34CLKSOURCE_LSE

LPUART1 Clock Source

RCC_LPUART1CLKSOURCE_PCLK3

RCC_LPUART1CLKSOURCE_SYSCLK

RCC_LPUART1CLKSOURCE_HSI

RCC_LPUART1CLKSOURCE_LSE

RCC_LPUART1CLKSOURCE_MSIK

Low Speed Clock Source

RCC_LSCOSOURCE_LSI

LSI selection for low speed clock output

RCC_LSCOSOURCE_LSE

LSE selection for low speed clock output

LTDC Clock Source

RCC_LTDCCLKSOURCE_PLL3

PLL3 divider R clock selected as LTDC kernel clock

RCC_LTDCCLKSOURCE_PLL2

PLL2 divider R clock selected as LTDC kernel clock

MDF1 Clock Source

RCC_MDF1CLKSOURCE_HCLK

RCC_MDF1CLKSOURCE_PLL1

RCC_MDF1CLKSOURCE_PLL3

RCC_MDF1CLKSOURCE_PIN

RCC_MDF1CLKSOURCE_MSIK

MSI Pll Mode Selection

RCC_MSIKPLL_MODE_SEL

PLL mode applied to MSIK (MSI kernel) clock output

RCC_MSISPLL_MODE_SEL

PLL mode applied to MSIS (MSI system) clock output

OctoSPI Clock Source

RCC_OSPICLKSOURCE_SYSCLK

RCC_OSPICLKSOURCE_MSIK

RCC_OSPICLKSOURCE_PLL1

RCC_OSPICLKSOURCE_PLL2

RCCEx Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_USART2

RCC_PERIPHCLK_USART3

RCC_PERIPHCLK_UART4

RCC_PERIPHCLK_UART5

RCC_PERIPHCLK_LPUART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2C2

RCC_PERIPHCLK_I2C3

RCC_PERIPHCLK_LPTIM1

RCC_PERIPHCLK_LPTIM2

RCC_PERIPHCLK_LPTIM34

RCC_PERIPHCLK_SAES

RCC_PERIPHCLK_SAI1

RCC_PERIPHCLK_SAI2

RCC_PERIPHCLK_ADCDAC

RCC_PERIPHCLK_MDF1

RCC_PERIPHCLK_ADF1

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_RNG

RCC_PERIPHCLK_ICLK

RCC_PERIPHCLK_SDMMC

RCC_PERIPHCLK_I2C4

RCC_PERIPHCLK_SPI1

RCC_PERIPHCLK_SPI2

RCC_PERIPHCLK_SPI3

RCC_PERIPHCLK_OSPI

RCC_PERIPHCLK_FDCAN1

RCC_PERIPHCLK_DAC1

RCC_PERIPHCLK_USART6

RCC_PERIPHCLK_I2C5

RCC_PERIPHCLK_I2C6

RCC_PERIPHCLK_HSPI

RCC_PERIPHCLK_LTDC

RCC_PERIPHCLK_DSI

RCC_PERIPHCLK_USBPHY

RCC_PERIPHCLOCK_ALL

RCCEx RNG Clock Source

RCC_RNGCLKSOURCE_HSI48

RCC_RNGCLKSOURCE_HSI48_DIV2

RCC_RNGCLKSOURCE_HSI

RCCEx SAES Clock Source

RCC_SAESCLKSOURCE_SHSI

RCC_SAESCLKSOURCE_SHSI_DIV2

SAI1 Clock Source

RCC_SAI1CLKSOURCE_PLL2

RCC_SAI1CLKSOURCE_PLL3

RCC_SAI1CLKSOURCE_PLL1

RCC_SAI1CLKSOURCE_PIN

RCC_SAI1CLKSOURCE_HSI

SAI2 Clock Source

RCC_SAI2CLKSOURCE_PLL2

RCC_SAI2CLKSOURCE_PLL3

RCC_SAI2CLKSOURCE_PLL1

RCC_SAI2CLKSOURCE_PIN

RCC_SAI2CLKSOURCE_HSI

SDMMC1/2 Clock Source

RCC_SDMMCCLKSOURCE_CLK48

RCC_SDMMCCLKSOURCE_PLL1

SPI1 Clock Source

RCC_SPI1CLKSOURCE_PCLK2

RCC_SPI1CLKSOURCE_SYSCLK

RCC_SPI1CLKSOURCE_HSI

RCC_SPI1CLKSOURCE_MSIK

SPI2 Clock Source

RCC_SPI2CLKSOURCE_PCLK1

RCC_SPI2CLKSOURCE_SYSCLK

RCC_SPI2CLKSOURCE_HSI

RCC_SPI2CLKSOURCE_MSIK

SPI3 Clock Source

RCC_SPI3CLKSOURCE_PCLK3

RCC_SPI3CLKSOURCE_SYSCLK

RCC_SPI3CLKSOURCE_HSI

RCC_SPI3CLKSOURCE_MSIK

UART4 Clock Source

RCC_UART4CLKSOURCE_PCLK1

RCC_UART4CLKSOURCE_SYSCLK

RCC_UART4CLKSOURCE_HSI

RCC_UART4CLKSOURCE_LSE

UART5 Clock Source

RCC_UART5CLKSOURCE_PCLK1

RCC_UART5CLKSOURCE_SYSCLK

RCC_UART5CLKSOURCE_HSI

RCC_UART5CLKSOURCE_LSE

USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK2

RCC_USART1CLKSOURCE_SYSCLK

RCC_USART1CLKSOURCE_HSI

RCC_USART1CLKSOURCE_LSE

USART2 Clock Source

RCC_USART2CLKSOURCE_PCLK1

RCC_USART2CLKSOURCE_SYSCLK

RCC_USART2CLKSOURCE_HSI

RCC_USART2CLKSOURCE_LSE

USART3 Clock Source

RCC_USART3CLKSOURCE_PCLK1

RCC_USART3CLKSOURCE_SYSCLK

RCC_USART3CLKSOURCE_HSI

RCC_USART3CLKSOURCE_LSE

USART6 Clock Source

RCC_USART6CLKSOURCE_PCLK1

RCC_USART6CLKSOURCE_SYSCLK

RCC_USART6CLKSOURCE_HSI

RCC_USART6CLKSOURCE_LSE

USB-PHY Clock Source

RCC_USBPHYCLKSOURCE_HSE

HSE clock selected as USBPHYC clock

RCC_USBPHYCLKSOURCE_HSE_DIV2

HSE clock divided by 2 selected as USBPHYC clock

RCC_USBPHYCLKSOURCE_PLL1

PLL1 divider P selected as USBPHYC clock

RCC_USBPHYCLKSOURCE_PLL1_DIV2

PLL1 divider P divided by 2 selected as USBPHYC clock

66 HAL RNG Generic Driver

66.1 RNG Firmware driver registers structures

66.1.1 RNG_InitTypeDef

RNG_InitTypeDef is defined in the stm32u5xx_hal_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  CED Clock error detection

66.1.2 __RNG_HandleTypeDef

__RNG_HandleTypeDef is defined in the stm32u5xx_hal_rng.h

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t RandomNumber***
- ***void(* ReadyDataCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RNG_TypeDef* __RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef __RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef __RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef __RNG_HandleTypeDef::State***
  RNG communication state
- ***__IO uint32_t __RNG_HandleTypeDef::ErrorCode***
  RNG Error code
- ***uint32_t __RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data
- ***void(* __RNG_HandleTypeDef::ReadyDataCallback)(struct __RNG_HandleTypeDef *hrng, uint32_t random32bit)***
  RNG Data Ready Callback
- ***void(* __RNG_HandleTypeDef::ErrorCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Error Callback
- ***void(* __RNG_HandleTypeDef::MspInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp Init callback
- ***void(* __RNG_HandleTypeDef::MspDeInitCallback)(struct __RNG_HandleTypeDef *hrng)***
  RNG Msp DeInit callback

66.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

66.2.1 How to use this driver

The RNG HAL driver can be used as follows:
1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 32 bit Random Number Generator contains a valid random data using (polling/interrupt) mode.
4. Get the 32 bit random number using HAL_RNG_GenerateRandomNumber() function.

66.2.2 Callback registration

The compilation define USE_HAL_RNG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_RNG_RegisterCallback() to register a user callback. Function HAL_RNG_RegisterCallback() allows to register following callbacks:
- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_RNG_UnRegisterCallback() to reset a callback to the default weak (overridden) function. HAL_RNG_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- ErrorCallback : RNG Error Callback.
- MspInitCallback : RNG MspInit.
- MspDeInitCallback : RNG MspDeInit.

For specific callback ReadyDataCallback, use dedicated register callbacks: respectively HAL_RNG_RegisterReadyDataCallback() , HAL_RNG_UnRegisterReadyDataCallback().

By default, after the HAL_RNG_Init() and when the state is HAL_RNG_STATE_RESET all callbacks are set to the corresponding weak (overridden) functions: example HAL_RNG_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak (overridden) functions in the HAL_RNG_Init() and HAL_RNG_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_RNG_Init() and HAL_RNG_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_RNG_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_RNG_STATE_READY or HAL_RNG_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_RNG_RegisterCallback() before calling HAL_RNG_DeInit() or HAL_RNG_Init() function.

When The compilation define USE_HAL_RNG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak (overridden) callbacks are used.

66.2.3 Initialization and configuration functions

This section provides functions allowing to:
- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle
- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:
- *HAL_RNG_Init()*
- *HAL_RNG_DeInit()*
- *HAL_RNG_MspInit()*
- *HAL_RNG_MspDeInit()*
- *HAL_RNG_RegisterCallback()*
- *HAL_RNG_UnRegisterCallback()*

- *HAL_RNG_RegisterReadyDataCallback()*
- *HAL_RNG_UnRegisterReadyDataCallback()*

66.2.4 Peripheral Control functions

This section provides functions allowing to:

- Get the 32 bit Random number
- Get the 32 bit Random number with interrupt enabled
- Handle RNG interrupt request

This section contains the following APIs:

- *HAL_RNG_GenerateRandomNumber()*
- *HAL_RNG_GenerateRandomNumber_IT()*
- *HAL_RNG_IRQHandler()*
- *HAL_RNG_ReadLastRandomNumber()*
- *HAL_RNG_ReadyDataCallback()*
- *HAL_RNG_ErrorCallback()*

66.2.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_RNG_GetState()*
- *HAL_RNG_GetError()*

66.2.6 Detailed description of functions

HAL_RNG_Init

Function name

HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG peripheral and creates the associated handle.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_DeInit

Function name

HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG peripheral.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_MspInit

Function name

void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_MspDeInit

Function name

void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID, pRNG_CallbackTypeDef pCallback)

Function description

Register a User RNG Callback To be used instead of the weak predefined callback.

Parameters

- **hrng:** RNG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_RNG_ERROR_CB_ID Error callback ID
  – HAL_RNG_MSPINIT_CB_ID MspInit callback ID
  – HAL_RNG_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_RNG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterCallback (RNG_HandleTypeDef * hrng, HAL_RNG_CallbackIDTypeDef CallbackID)

Function description

Unregister an RNG Callback RNG callback is redirected to the weak predefined callback.

Parameters

- **hrng:** RNG handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_RNG_ERROR_CB_ID Error callback ID
  - HAL_RNG_MSPINIT_CB_ID MspInit callback ID
  - HAL_RNG_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_RNG_RegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_RegisterReadyDataCallback (RNG_HandleTypeDef * hrng, pRNG_ReadyDataCallbackTypeDef pCallback)

Function description

Register Data Ready RNG Callback To be used instead of the weak HAL_RNG_ReadyDataCallback() predefined callback.

Parameters

- **hrng:** RNG handle
- **pCallback:** pointer to the Data Ready Callback function

Return values

- **HAL:** status

HAL_RNG_UnRegisterReadyDataCallback

Function name

HAL_StatusTypeDef HAL_RNG_UnRegisterReadyDataCallback (RNG_HandleTypeDef * hrng)

Function description

UnRegister the Data Ready RNG Callback Data Ready RNG Callback is redirected to the weak HAL_RNG_ReadyDataCallback() predefined callback.

Parameters

- **hrng:** RNG handle

Return values

- **HAL:** status

HAL_RNG_GenerateRandomNumber

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random32bit)

Function description

Generates a 32-bit random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** pointer to generated random number variable if successful.

Return values

- **HAL:** status

Notes

- This function checks value of RNG_FLAG_DRDY flag to know if valid random number is available in the DR register (RNG_FLAG_DRDY flag set whenever a random number is available through the RNG_DR register). After transitioning from 0 to 1 (random number available), RNG_FLAG_DRDY flag remains high until output buffer becomes empty after reading four words from the RNG_DR register, i.e. further function calls will immediately return a new u32 random number (additional words are available and can be read by the application, till RNG_FLAG_DRDY flag remains high).
- When no more random number data is available in DR register, RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_GenerateRandomNumber_IT

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber_IT (RNG_HandleTypeDef * hrng)

Function description

Generates a 32-bit random number in interrupt mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (const RNG_HandleTypeDef * hrng)

Function description

Read latest generated random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **random:** value

HAL_RNG_IRQHandler

Function name

void HAL_RNG_IRQHandler (RNG_HandleTypeDef * hrng)

Function description

Handles RNG interrupt request.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

Notes

- In the case of a clock error, the RNG is no more able to generate random numbers because the PLL48CLK clock is not correct. User has to check that the clock controller is correctly configured to provide the RNG clock and clear the CEIS bit using __HAL_RNG_CLEAR_IT(). The clock error has no impact on the previously generated random numbers, and the RNG_DR register contents can be used.
- In the case of a seed error, the generation of random numbers is interrupted as long as the SECS bit is '1'. If a number is available in the RNG_DR register, it must not be used because it may not have enough entropy. In this case, it is recommended to clear the SEIS bit using __HAL_RNG_CLEAR_IT(), then disable and enable the RNG peripheral to reinitialize and restart the RNG.
- User-written HAL_RNG_ErrorCallback() API is called once whether SEIS or CEIS are set.

HAL_RNG_ErrorCallback

Function name

void HAL_RNG_ErrorCallback (RNG_HandleTypeDef * hrng)

Function description

RNG error callbacks.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_ReadyDataCallback

Function name

void HAL_RNG_ReadyDataCallback (RNG_HandleTypeDef * hrng, uint32_t random32bit)

Function description

Data Ready callback in non-blocking mode.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random32bit:** generated random number.

Return values

- **None:**

Notes

- When RNG_FLAG_DRDY flag value is set, first random number has been read from DR register in IRQ Handler and is provided as callback parameter. Depending on valid data available in the conditioning output buffer, additional words can be read by the application from DR register till DRDY bit remains high.

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (const RNG_HandleTypeDef * hrng)

Function description

Returns the RNG state.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** state

HAL_RNG_GetError

Function name

uint32_t HAL_RNG_GetError (const RNG_HandleTypeDef * hrng)

Function description

Return the RNG handle error code.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **RNG:** Error Code

RNG_RecoverSeedError

Function name

HAL_StatusTypeDef RNG_RecoverSeedError (RNG_HandleTypeDef * hrng)

Function description

RNG sequence to recover from a seed error.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **HAL:** status

66.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

66.3.1 RNG

RNG

RNG Error Definition

HAL_RNG_ERROR_NONE

No error

HAL_RNG_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_RNG_ERROR_TIMEOUT

Timeout error

HAL_RNG_ERROR_BUSY

Busy error

HAL_RNG_ERROR_SEED

Seed error

HAL_RNG_ERROR_CLOCK

Clock error

RNG Interrupt definition

RNG_IT_DRDY

Data Ready interrupt

RNG_IT_CEI

Clock error interrupt

RNG_IT_SEI

Seed error interrupt

RNG Flag definition

RNG_FLAG_DRDY

Data ready

RNG_FLAG_CECS

Clock error current status

RNG_FLAG_SECS

Seed error current status

RNG Clock Error Detection

RNG_CED_ENABLE

Clock error detection Enabled

RNG_CED_DISABLE

Clock error detection Disabled

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:

- Reset RNG handle state.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_ENABLE

Description:

- Enables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE

Description:

- Disables the RNG peripheral.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_FLAG

Description:

- Check the selected RNG flag status.

Parameters:

- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  – RNG_FLAG_DRDY: Data ready
  – RNG_FLAG_CECS: Clock error current status
  – RNG_FLAG_SECS: Seed error current status

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_RNG_CLEAR_FLAG

Description:

- Clears the selected RNG flag status.

Parameters:

- __HANDLE__: RNG handle
- __FLAG__: RNG flag to clear

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment, flags RNG_FLAG_DRDY, RNG_FLAG_CECS and RNG_FLAG_SECS are read-only.

__HAL_RNG_ENABLE_IT

Description:

- Enables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_DISABLE_IT

Description:

- Disables the RNG interrupts.

Parameters:

- __HANDLE__: RNG Handle

Return value:

- None

__HAL_RNG_GET_IT

Description:

- Checks whether the specified RNG interrupt has occurred or not.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to check. This parameter can be one of the following values:
  - RNG_IT_DRDY: Data ready interrupt
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_RNG_CLEAR_IT

Description:

- Clear the RNG interrupt status flags.

Parameters:

- __HANDLE__: RNG Handle
- __INTERRUPT__: specifies the RNG interrupt status flag to clear. This parameter can be one of the following values:
  - RNG_IT_CEI: Clock error interrupt
  - RNG_IT_SEI: Seed error interrupt

Return value:

- None

Notes:

- RNG_IT_DRDY flag is read-only, reading RNG_DR register automatically clears RNG_IT_DRDY.

Auto Reset configuration

RNG_ARDIS_ENABLE

automatic reset after seed error

RNG_ARDIS_DISABLE

Disable automatic reset after seed error

Value used to configure an internal

RNG_CLKDIV_BY_1

No clock division

RNG_CLKDIV_BY_2

2 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_4

4 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_8

8 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_16

16 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_32

32 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_64

64 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_128

128 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_256

256 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_512

512 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_1024

1024 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_2048

2048 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_4096

4096 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_8192

8192 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_16384

16384 RNG clock cycles per internal RNG clock

RNG_CLKDIV_BY_32768

32768 RNG clock cycles per internal RNG clock

NIST Compliance configuration

RNG_NIST_COMPLIANT

NIST compliant configuration

RNG_CUSTOM_NIST

Custom NIST configuration

67 HAL RNG Extension Driver

67.1 RNGEx Firmware driver registers structures

67.1.1 RNG_ConfigTypeDef

RNG_ConfigTypeDef is defined in the stm32u5xx_hal_rng_ex.h

Data Fields

- ***uint32_t Config1***
- ***uint32_t Config2***
- ***uint32_t Config3***
- ***uint32_t ClockDivider***
- ***uint32_t NistCompliance***
- ***uint32_t AutoReset***
- ***uint32_t HealthTest***
- ***uint32_t NoiseSource***

Field Documentation

- ***uint32_t RNG_ConfigTypeDef::Config1***
  Config1 must be a value between 0 and 0x3F
- ***uint32_t RNG_ConfigTypeDef::Config2***
  Config2 must be a value between 0 and 0x7
- ***uint32_t RNG_ConfigTypeDef::Config3***
  Config3 must be a value between 0 and 0xF
- ***uint32_t RNG_ConfigTypeDef::ClockDivider***
  Clock Divider factor.This parameter can be a value of ***RNG_Ex_Clock_Divider_Factor***
- ***uint32_t RNG_ConfigTypeDef::NistCompliance***
  NIST compliance.This parameter can be a value of ***RNG_Ex_NIST_Compliance***
- ***uint32_t RNG_ConfigTypeDef::AutoReset***
  automatic reset When a noise source error occurs value of ***RNG_Ex_Auto_Reset***
- ***uint32_t RNG_ConfigTypeDef::HealthTest***
  RNG health test control must be a value between 0x0FFCABFF and 0x00005200
- ***uint32_t RNG_ConfigTypeDef::NoiseSource***
  RNG noise source control(Oscillator Enable signals) must be a value between 0x0 and 0x0003FFFF

67.2 RNGEx Firmware driver defines

The following section lists the various define and macros of the module.

67.2.1 RNGEx

RNGEx

68 HAL RTC Generic Driver

68.1 RTC Firmware driver registers structures

68.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32u5xx_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutRemap***
- ***uint32_t OutPutPolarity***
- ***uint32_t OutPutType***
- ***uint32_t OutPutPullUp***
- ***uint32_t BinMode***
- ***uint32_t BinMixBcdU***

Field Documentation

- ***uint32_t RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hour Format. This parameter can be a value of ***RTC_Hour_Formats***
- ***uint32_t RTC_InitTypeDef::AsynchPrediv***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- ***uint32_t RTC_InitTypeDef::SynchPrediv***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- ***uint32_t RTC_InitTypeDef::OutPut***
  Specifies which signal will be routed to the RTC output. This parameter can be a value of ***RTCEx_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutRemap***
  Specifies the remap for RTC output. This parameter can be a value of ***RTC_Output_ALARM_OUT_Remap***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutType***
  Specifies the RTC Output Pin mode. This parameter can be a value of ***RTC_Output_Type_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::OutPutPullUp***
  Specifies the RTC Output Pull-Up mode. This parameter can be a value of ***RTC_Output_PullUp_ALARM_OUT***
- ***uint32_t RTC_InitTypeDef::BinMode***
  Specifies the RTC binary mode. This parameter can be a value of ***RTCEx_Binary_Mode***
- ***uint32_t RTC_InitTypeDef::BinMixBcdU***
  Specifies the BCD calendar update if and only if BinMode = RTC_BINARY_MIX. This parameter can be a value of ***RTCEx_Binary_mix_BCDU***

68.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32u5xx_hal_rtc.h

Data Fields

- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

- ***uint8_t TimeFormat***
- ***uint32_t SubSeconds***
- ***uint32_t SecondFraction***
- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hour. This parameter must be a number between: Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between: Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected
- ***uint8_t RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59
- ***uint8_t RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_AM_PM_Definitions***
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
  Specifies the RTC_SSR RTC Sub Second register content. This field is not used by HAL_RTC_SetTime. If the free running 32 bit counter is not activated (mode binary none)
  – This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity else
  – This parameter corresponds to the free running 32 bit counter.
- ***uint32_t RTC_TimeTypeDef::SecondFraction***
  Specifies the range or granularity of Sub Second register content corresponding to Synchronous pre-scaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
  This interface is deprecated. To manage Daylight SavingTime,use HAL_RTC_DST_xxx functions
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

68.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32u5xx_hal_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Date***
- ***uint8_t Year***

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint8_t RTC_DateTypeDef::Month***
  Specifies the RTC Date Month (in BCD format). This parameter can be a value of ***RTC_Month_Date_Definitions***
- ***uint8_t RTC_DateTypeDef::Date***
  Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

68.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32u5xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t BinaryAutoClr***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t FlagAutoClr***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_AlarmMask_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
  Specifies the RTC Alarm SubSeconds Masks. if Binary mode is RTC_BINARY_ONLY or is RTC_BINARY_MIX This parameter can be a value of ***RTCEx_Alarm_Sub_Seconds_binary_Masks_Definitions*** else if Binary mode is RTC_BINARY_NONE This parameter can be a value of ***RTC_Alarm_Sub_Seconds_Masks_Definitions***
- ***uint32_t RTC_AlarmTypeDef::BinaryAutoClr***
  Clear synchronously counter (RTC_SSR) on binary alarm. RTC_ALARMSUBSECONDBIN_AUTOCLR_YES must only be used if Binary mode is RTC_BINARY_ONLY This parameter can be a value of ***RTCEx_Alarm_Sub_Seconds_binary_Clear_Definitions***
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of ***RTC_AlarmDateWeekDay_Definitions***
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of ***RTC_WeekDay_Definitions***
- ***uint32_t RTC_AlarmTypeDef::FlagAutoClr***
  Specifies the alarm trigger generation. This feature is meaningful to avoid any RTC software execution after configuration. This parameter can be a value of ***RTC_ALARM_Flag_AutoClear_Definitions***
- ***uint32_t RTC_AlarmTypeDef::Alarm***
  Specifies the alarm . This parameter can be a value of ***RTC_Alarms_Definitions***

68.1.5 __RTC_HandleTypeDef

__RTC_HandleTypeDef is defined in the stm32u5xx_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***
- ***void(* AlarmAEventCallback***
- ***void(* AlarmBEventCallback***
- ***void(* TimeStampEventCallback***
- ***void(* WakeUpTimerEventCallback***
- ***void(* SSRUEventCallback***
- ***void(* Tamper1EventCallback***
- ***void(* Tamper2EventCallback***
- ***void(* Tamper3EventCallback***
- ***void(* Tamper4EventCallback***

- ***void(* Tamper5EventCallback***
- ***void(* Tamper6EventCallback***
- ***void(* Tamper7EventCallback***
- ***void(* Tamper8EventCallback***
- ***void(* InternalTamper1EventCallback***
- ***void(* InternalTamper2EventCallback***
- ***void(* InternalTamper3EventCallback***
- ***void(* InternalTamper5EventCallback***
- ***void(* InternalTamper6EventCallback***
- ***void(* InternalTamper7EventCallback***
- ***void(* InternalTamper8EventCallback***
- ***void(* InternalTamper9EventCallback***
- ***void(* InternalTamper11EventCallback***
- ***void(* InternalTamper12EventCallback***
- ***void(* InternalTamper13EventCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RTC_TypeDef* __RTC_HandleTypeDef::Instance***
  Legacy register base address. Not used anymore, the driver directly uses cmsis base address
- ***RTC_InitTypeDef __RTC_HandleTypeDef::Init***
  RTC required parameters
- ***HAL_LockTypeDef __RTC_HandleTypeDef::Lock***
  RTC locking object
- ***__IO HAL_RTCStateTypeDef __RTC_HandleTypeDef::State***
  Time communication state
- ***void(* __RTC_HandleTypeDef::AlarmAEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm A Event callback
- ***void(* __RTC_HandleTypeDef::AlarmBEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Alarm B Event callback
- ***void(* __RTC_HandleTypeDef::TimeStampEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC TimeStamp Event callback
- ***void(* __RTC_HandleTypeDef::WakeUpTimerEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC WakeUpTimer Event callback
- ***void(* __RTC_HandleTypeDef::SSRUEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC SSRU Event callback
- ***void(* __RTC_HandleTypeDef::Tamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 1 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper4EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 4 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper5EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 5 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper6EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 6 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper7EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 7 Event callback
- ***void(* __RTC_HandleTypeDef::Tamper8EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Tamper 8 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper1EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 1 Event callback

- ***void(* __RTC_HandleTypeDef::InternalTamper2EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 2 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper3EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 3 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper5EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 5 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper6EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 6 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper7EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 7 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper8EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 8 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper9EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 9 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper11EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 11 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper12EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 12 Event callback
- ***void(* __RTC_HandleTypeDef::InternalTamper13EventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Internal Tamper 13 Event callback
- ***void(* __RTC_HandleTypeDef::MspInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp Init callback
- ***void(* __RTC_HandleTypeDef::MspDeInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp DeInit callback

68.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

68.2.1 RTC Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

68.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values. A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.
3. Tamper detection event resets all data backup registers.

68.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the __HAL_RCC_PWR_CLK_ENABLE() function.
- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.
- Select the RTC clock source using the __HAL_RCC_RTC_CONFIG() function.
- Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() function.

To enable access to the RTC Domain and RTC registers, proceed as follows:

1. Call the function HAL_RCCEx_PeriphCLKConfig with RCC_PERIPHCLK_RTC for PeriphClockSelection and select RTCClockSelection (LSE, LSI or HSEdiv32)
2. Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() macro.

68.2.4 How to use RTC Driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.

Alarm configuration

- To configure the RTC Alarm use the HAL_RTC_SetAlarm() function. You can also configure the RTC Alarm with interrupt mode using the HAL_RTC_SetAlarm_IT() function.
- To read the RTC Alarm, use the HAL_RTC_GetAlarm() function.

68.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

Callback registration

68.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   – A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
   – When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- *HAL_RTC_Init()*
- *HAL_RTC_DeInit()*
- *HAL_RTC_RegisterCallback()*
- *HAL_RTC_UnRegisterCallback()*
- *HAL_RTC_MspInit()*
- *HAL_RTC_MspDeInit()*

68.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:
- *HAL_RTC_SetTime()*
- *HAL_RTC_DST_Add1Hour()*
- *HAL_RTC_DST_Sub1Hour()*
- *HAL_RTC_DST_SetStoreOperation()*
- *HAL_RTC_DST_ClearStoreOperation()*
- *HAL_RTC_DST_ReadStoreOperation()*
- *HAL_RTC_GetTime()*
- *HAL_RTC_SetDate()*
- *HAL_RTC_GetDate()*

68.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:
- *HAL_RTC_SetAlarm()*
- *HAL_RTC_SetAlarm_IT()*
- *HAL_RTC_DeactivateAlarm()*
- *HAL_RTC_GetAlarm()*
- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

68.2.9 Peripheral Control functions

This subsection provides functions allowing to
- Wait for RTC Time and Date Synchronization

This section contains the following APIs:
- *HAL_RTC_WaitForSynchro()*

68.2.10 Peripheral State functions

This subsection provides functions allowing to
- Get RTC state

This section contains the following APIs:
- *HAL_RTC_GetState()*

68.2.11 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC peripheral.

Parameters
- **hrtc:** RTC handle

Return values
- **HAL:** status

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC peripheral.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This function does not reset the RTC Backup Data registers.

HAL_RTC_MspInit

Function name

void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)

Function description

Initialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_MspDeInit

Function name

void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitialize the RTC MSP.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_RegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID, pRTC_CallbackTypeDef pCallback)

Function description

Register a User RTC Callback To be used instead of the weak predefined callback.

Parameters

- **hrtc:** RTC handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  – HAL_RTC_ALARM_B_EVENT_CB_ID Alarm B Event Callback ID
  – HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event Callback ID
  – HAL_RTC_WAKEUPTIMER_EVENT_CB_ID WakeUp Timer Event Callback ID
  – HAL_RTC_TAMPER1_EVENT_CB_ID Tamper 1 Callback ID
  – HAL_RTC_TAMPER2_EVENT_CB_ID Tamper 2 Callback ID
  – HAL_RTC_TAMPER3_EVENT_CB_ID Tamper 3 Callback ID
  – HAL_RTC_TAMPER4_EVENT_CB_ID Tamper 4 Callback ID
  – HAL_RTC_TAMPER5_EVENT_CB_ID Tamper 5 Callback ID
  – HAL_RTC_TAMPER6_EVENT_CB_ID Tamper 6 Callback ID
  – HAL_RTC_TAMPER7_EVENT_CB_ID Tamper 7 Callback ID
  – HAL_RTC_TAMPER8_EVENT_CB_ID Tamper 8 Callback ID
  – HAL_RTC_INTERNAL_TAMPER1_EVENT_CB_ID Internal Tamper 1 Callback ID
  – HAL_RTC_INTERNAL_TAMPER2_EVENT_CB_ID Internal Tamper 2 Callback ID
  – HAL_RTC_INTERNAL_TAMPER3_EVENT_CB_ID Internal Tamper 3 Callback ID
  – HAL_RTC_INTERNAL_TAMPER5_EVENT_CB_ID Internal Tamper 5 Callback ID
  – HAL_RTC_INTERNAL_TAMPER6_EVENT_CB_ID Internal Tamper 6 Callback ID
  – HAL_RTC_INTERNAL_TAMPER7_EVENT_CB_ID Internal Tamper 7 Callback ID
  – HAL_RTC_INTERNAL_TAMPER8_EVENT_CB_ID Internal Tamper 8 Callback ID
  – HAL_RTC_INTERNAL_TAMPER9_EVENT_CB_ID Internal Tamper 9 Callback ID
  – HAL_RTC_INTERNAL_TAMPER11_EVENT_CB_ID Internal Tamper 11 Callback ID
  – HAL_RTC_INTERNAL_TAMPER12_EVENT_CB_ID Internal Tamper 12 Callback ID
  – HAL_RTC_INTERNAL_TAMPER13_EVENT_CB_ID Internal Tamper 13 Callback ID
  – HAL_RTC_MSPINIT_CB_ID Msp Init callback ID
  – HAL_RTC_MSPDEINIT_CB_ID Msp DeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_RTC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_RTC_UnRegisterCallback (RTC_HandleTypeDef * hrtc, HAL_RTC_CallbackIDTypeDef CallbackID)

Function description

Unregister an RTC Callback RTC callback is redirected to the weak predefined callback.

Parameters

- **hrtc:** RTC handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  - HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  - HAL_RTC_ALARM_B_EVENT_CB_ID Alarm B Event Callback ID
  - HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event Callback ID
  - HAL_RTC_SSRU_EVENT_CB_ID SSRU Callback ID
  - HAL_RTC_WAKEUPTIMER_EVENT_CB_ID WakeUp Timer Event Callback ID
  - HAL_RTC_TAMPER1_EVENT_CB_ID Tamper 1 Callback ID
  - HAL_RTC_TAMPER2_EVENT_CB_ID Tamper 2 Callback ID
  - HAL_RTC_TAMPER3_EVENT_CB_ID Tamper 3 Callback ID
  - HAL_RTC_TAMPER4_EVENT_CB_ID Tamper 4 Callback ID
  - HAL_RTC_TAMPER5_EVENT_CB_ID Tamper 5 Callback ID
  - HAL_RTC_TAMPER6_EVENT_CB_ID Tamper 6 Callback ID
  - HAL_RTC_TAMPER7_EVENT_CB_ID Tamper 7 Callback ID
  - HAL_RTC_TAMPER8_EVENT_CB_ID Tamper 8 Callback ID
  - HAL_RTC_INTERNAL_TAMPER1_EVENT_CB_ID Internal Tamper 1 Callback ID
  - HAL_RTC_INTERNAL_TAMPER2_EVENT_CB_ID Internal Tamper 2 Callback ID
  - HAL_RTC_INTERNAL_TAMPER3_EVENT_CB_ID Internal Tamper 3 Callback ID
  - HAL_RTC_INTERNAL_TAMPER5_EVENT_CB_ID Internal Tamper 5 Callback ID
  - HAL_RTC_INTERNAL_TAMPER6_EVENT_CB_ID Internal Tamper 6 Callback ID
  - HAL_RTC_INTERNAL_TAMPER7_EVENT_CB_ID Internal Tamper 7 Callback ID
  - HAL_RTC_INTERNAL_TAMPER8_EVENT_CB_ID Internal Tamper 8 Callback ID
  - HAL_RTC_INTERNAL_TAMPER9_EVENT_CB_ID Internal Tamper 9 Callback ID
  - HAL_RTC_INTERNAL_TAMPER11_EVENT_CB_ID Internal Tamper 11 Callback ID
  - HAL_RTC_INTERNAL_TAMPER12_EVENT_CB_ID Internal Tamper 12 Callback ID
  - HAL_RTC_INTERNAL_TAMPER13_EVENT_CB_ID Internal Tamper 13 Callback ID
  - HAL_RTC_MSPINIT_CB_ID Msp Init callback ID
  - HAL_RTC_MSPDEINIT_CB_ID Msp DeInit callback ID

Return values

- **HAL:** status

HAL_RTC_SetTime

Function name

HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Set RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** Pointer to Time structure if Binary mode is RTC_BINARY_ONLY, this parameter is not used and RTC_SSR will be automatically reset to 0xFFFFFFFF else sTime->SubSeconds is not used and RTC_SSR will be automatically reset to the A 7-bit async prescaler (RTC_PRER_PREDIV_A)
- **Format:** Format of sTime->Hours, sTime->Minutes and sTime->Seconds. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values
  - RTC_FORMAT_BIN: Binary format
  - RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_GetTime

Function name

HAL_StatusTypeDef HAL_RTC_GetTime (const RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** if Binary mode is RTC_BINARY_ONLY, sTime->SubSeconds only is updated else Pointer to Time structure with Hours, Minutes and Seconds fields returned with input format (BIN or BCD), also SubSeconds field returning the RTC_SSR register content and SecondFraction field the Synchronous pre-scaler factor to be used for second fraction ratio computation.
- **Format:** Format of sTime->Hours, sTime->Minutes and sTime->Seconds. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values:
  – RTC_FORMAT_BIN: Binary format
  – RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

Notes

- You can use SubSeconds and SecondFraction (sTime structure fields returned) to convert SubSeconds value in second fraction ratio with time unit following generic formula: Second fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS
- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read to ensure consistency between the time and date values.

HAL_RTC_SetDate

Function name

HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Set RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to date structure
- **Format:** Format of sDate->Year, sDate->Month and sDate->Weekday. This parameter can be one of the following values:
  – RTC_FORMAT_BIN: Binary format
  – RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_GetDate

Function name

HAL_StatusTypeDef HAL_RTC_GetDate (const RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Get RTC current date.

Parameters

- **hrtc:** RTC handle
- **sDate:** Pointer to Date structure
- **Format:** Format of sDate->Year, sDate->Month and sDate->Weekday. This parameter can be one of the following values:
  – RTC_FORMAT_BIN: Binary format
  – RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

Notes

- You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

HAL_RTC_DST_Add1Hour

Function name

void HAL_RTC_DST_Add1Hour (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Add one hour to the calendar in one single operation without going through the initialization procedure.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_DST_Sub1Hour

Function name

void HAL_RTC_DST_Sub1Hour (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Subtract one hour from the calendar in one single operation without going through the initialization procedure.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_DST_SetStoreOperation

Function name

void HAL_RTC_DST_SetStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Set the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

Notes

- It can be used by the software in order to memorize the DST status.

HAL_RTC_DST_ClearStoreOperation

Function name

void HAL_RTC_DST_ClearStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Clear the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_DST_ReadStoreOperation

Function name

uint32_t HAL_RTC_DST_ReadStoreOperation (const RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, Read the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **operation:** see RTC_StoreOperation_Definitions

HAL_RTC_SetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure if Binary mode is RTC_BINARY_ONLY, 3 fields only are used sAlarm->AlarmTime.SubSeconds sAlarm->AlarmSubSecondMask sAlarm->BinaryAutoClr
- **Format:** of the entered parameters. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values
  – RTC_FORMAT_BIN: Binary format
  – RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_SetAlarm_IT

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Set the specified RTC Alarm with Interrupt.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Alarm structure if Binary mode is RTC_BINARY_ONLY, 3 fields only are used sAlarm->AlarmTime.SubSeconds sAlarm->AlarmSubSecondMask sAlarm->BinaryAutoClr
- **Format:** Specifies the format of the entered parameters. if Binary mode is RTC_BINARY_ONLY, this parameter is not used else this parameter can be one of the following values
  – RTC_FORMAT_BIN: Binary format
  – RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_DeactivateAlarm

Function name

HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function description

Deactivate the specified RTC Alarm.

Parameters

- **hrtc:** RTC handle
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  – RTC_ALARM_A: AlarmA
  – RTC_ALARM_B: AlarmB

Return values

- **HAL:** status

HAL_RTC_GetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_GetAlarm (const RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)

Function description

Get the RTC Alarm value and masks.

Parameters

- **hrtc:** RTC handle
- **sAlarm:** Pointer to Date structure
- **Alarm:** Specifies the Alarm. This parameter can be one of the following values:
  – RTC_ALARM_A: AlarmA
  – RTC_ALARM_B: AlarmB
- **Format:** Specifies the format of the entered parameters. This parameter can be one of the following values:
  – RTC_FORMAT_BIN: Binary format
  – RTC_FORMAT_BCD: BCD format

Return values

- **HAL:** status

HAL_RTC_AlarmIRQHandler

Function name

void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Alarm secure interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle AlarmA Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTC_AlarmAEventCallback

Function name

void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm A secure secure callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_WaitForSynchro

Function name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

Wait until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Resynchronization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

HAL_RTC_GetState

Function name

HAL_RTCStateTypeDef HAL_RTC_GetState (const RTC_HandleTypeDef * hrtc)

Function description

Return the RTC handle state.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** state

RTC_EnterInitMode

Function name

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

Enter the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The RTC Initialization mode is write protected, use the __HAL_RTC_WRITEPROTECTION_DISABLE() before calling this function.

RTC_ExitInitMode

Function name

HAL_StatusTypeDef RTC_ExitInitMode (RTC_HandleTypeDef * hrtc)

Function description

Exit the RTC Initialization mode.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

RTC_ByteToBcd2

Function name

uint8_t RTC_ByteToBcd2 (uint8_t Value)

Function description

Convert a 2 digit decimal to BCD format.

Parameters

- **Value:** Byte to be converted

Return values

- **Converted:** byte

RTC_Bcd2ToByte

Function name

uint8_t RTC_Bcd2ToByte (uint8_t Value)

Function description

Convert from 2 digit BCD to Binary.

Parameters

- **Value:** BCD value to be converted

Return values

- **Converted:** word

68.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

68.3.1 RTC

RTC

RTC AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC AlarmMask Definitions

RTC_ALARMMASK_NONE

RTC_ALARMMASK_DATEWEEKDAY

RTC_ALARMMASK_HOURS

RTC_ALARMMASK_MINUTES

RTC_ALARMMASK_SECONDS

RTC_ALARMMASK_ALL

RTC Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

RTC Alarms Flag Auto Clear Definitions

ALARM_FLAG_AUTOCLR_ENABLE

ALARM_FLAG_AUTOCLR_DISABLE

RTC Alarm Sub Seconds Masks Definitions

RTC_ALARMSUBSECONDMASK_ALL

All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:1] not used in Alarmcomparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:2] not used in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:3] not used in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:4] not used in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:5] not used in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:6] not used in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:7] not used in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:8] not used in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:9] not used in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:10] not used in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:11] not used in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:12] not used in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14:13] not used in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDMASK_SS14

SS[14] not used in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDMASK_NONE

SS[14:0] are compared and must match to activate alarm.

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC Clear Flags Definitions

RTC_CLEAR_SSRUF

Clear SSR underflow flag

RTC_CLEAR_ITSF

Clear Internal Time-stamp flag

RTC_CLEAR_TSOVF

Clear Time-stamp overflow flag

RTC_CLEAR_TSF

Clear Time-stamp flag

RTC_CLEAR_WUTF

Clear Wakeup timer flag

RTC_CLEAR_ALRBF

Clear Alarm B flag

RTC_CLEAR_ALRAF

Clear Alarm A flag

RTC DayLightSaving Definitions

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:
- Reset RTC handle state.

Parameters:
- __HANDLE__: RTC handle.

Return value:
- None

__HAL_RTC_WRITEPROTECTION_DISABLE

Description:
- Disable the write protection for RTC registers.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_WRITEPROTECTION_ENABLE

Description:
- Enable the write protection for RTC registers.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_DAYLIGHT_SAVING_TIME_ADD1H

Description:
- Add 1 hour (summer time change).

Parameters:
- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:
- None

Notes:
- This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

__HAL_RTC_DAYLIGHT_SAVING_TIME_SUB1H

Description:
- Subtract 1 hour (winter time change).

Parameters:
- __HANDLE__: specifies the RTC handle.
- __BKP__: Backup This parameter can be:
  - RTC_STOREOPERATION_RESET
  - RTC_STOREOPERATION_SET

Return value:
- None

Notes:
- This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

__HAL_RTC_ALARMA_ENABLE

Description:
- Enable the RTC ALARMA peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_ALARMA_DISABLE

Description:
- Disable the RTC ALARMA peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_ALARMB_ENABLE

Description:
- Enable the RTC ALARMB peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_ALARMB_DISABLE

Description:
- Disable the RTC ALARMB peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_ALARM_ENABLE_IT

Description:
- Enable the RTC Alarm interrupt.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:
- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- None

__HAL_RTC_ALARM_GET_IT

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_ALARM_GET_IT_SOURCE

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
  - RTC_IT_ALRA Alarm A interrupt
  - RTC_IT_ALRB Alarm B interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_ALARM_GET_FLAG

Description:

- Get the selected RTC Alarms flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_RTC_ALARM_CLEAR_FLAG

Description:

- Clear the RTC Alarms pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
  - RTC_FLAG_ALRAF
  - RTC_FLAG_ALRBF

Return value:

- None

__HAL_RTC_IS_CALENDAR_INITIALIZED

Description:

- Check whether if the RTC Calendar is initialized.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- The: state of RTC Calendar initialization (TRUE or FALSE).

RTC Flags Definitions

RTC_FLAG_RECALPF

Recalibration pending Flag

RTC_FLAG_INITF

Initialization flag

RTC_FLAG_RSF

Registers synchronization flag

RTC_FLAG_INITS

Initialization status flag

RTC_FLAG_SHPF

Shift operation pending flag

RTC_FLAG_WUTWF

Wakeup timer write flag

RTC_FLAG_SSRUF

Clear SSR underflow flag

RTC_FLAG_ITSF

Clear Internal Time-stamp flag

RTC_FLAG_TSOVF

Clear Time-stamp overflow flag

RTC_FLAG_TSF

Clear Time-stamp flag

RTC_FLAG_WUTF

Clear Wakeup timer flag

RTC_FLAG_ALRBF

Clear Alarm B flag

RTC_FLAG_ALRAF

Clear Alarm A flag

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interruptions Flag Mask

RTC_FLAG_MASK

RTC flags mask

RTC Interrupts Definitions

RTC_IT_TS

Enable Timestamp Interrupt

RTC_IT_WUT

Enable Wakeup timer Interrupt

RTC_IT_SSRU

Enable SSR Underflow Interrupt

RTC_IT_ALRA

Enable Alarm A Interrupt

RTC_IT_ALRB

Enable Alarm B Interrupt

RTC Private macros to check input parameters

IS_RTC_OUTPUT

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT_POL

IS_RTC_OUTPUT_TYPE

IS_RTC_OUTPUT_PULLUP

IS_RTC_OUTPUT_REMAP

IS_RTC_HOURFORMAT12

IS_RTC_DAYLIGHT_SAVING

IS_RTC_STORE_OPERATION

IS_RTC_FORMAT

IS_RTC_YEAR

IS_RTC_MONTH

IS_RTC_DATE

IS_RTC_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

IS_RTC_ALARM_MASK

IS_RTC_ALARM

IS_RTC_ALARM_SUB_SECOND_VALUE

IS_RTC_ALARM_SUB_SECOND_MASK

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

RTC Month Date Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

RTC Output ALARM OUT Remap

RTC_OUTPUT_REMAP_NONE

RTC_OUTPUT_REMAP_POS1

RTC Output Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Pull-Up ALARM OUT

RTC_OUTPUT_PULLUP_NONE

RTC_OUTPUT_PULLUP_ON

RTC Output Type ALARM OUT

RTC_OUTPUT_TYPE_PUSHPULL

RTC_OUTPUT_TYPE_OPENDRAIN

RTC StoreOperation Definitions

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

RTC WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

69 HAL RTC Extension Driver

69.1 RTCEx Firmware driver registers structures

69.1.1 RTC_TamperTypeDef

RTC_TamperTypeDef is defined in the stm32u5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Tamper***
- ***uint32_t Trigger***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***
- ***uint32_t Filter***
- ***uint32_t SamplingFrequency***
- ***uint32_t PrechargeDuration***
- ***uint32_t TamperPullUp***
- ***uint32_t TimeStampOnTamperDetection***

Field Documentation

- ***uint32_t RTC_TamperTypeDef::Tamper***
  Specifies the Tamper Pin. This parameter can be a value of ***RTCEx_Tamper_Pins***
- ***uint32_t RTC_TamperTypeDef::Trigger***
  Specifies the Tamper Trigger. This parameter can be a value of ***RTCEx_Tamper_Trigger***
- ***uint32_t RTC_TamperTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_TamperTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***
- ***uint32_t RTC_TamperTypeDef::Filter***
  Specifies the TAMP Filter Tamper. This parameter can be a value of ***RTCEx_Tamper_Filter***
- ***uint32_t RTC_TamperTypeDef::SamplingFrequency***
  Specifies the sampling frequency. This parameter can be a value of ***RTCEx_Tamper_Sampling_Frequencies***
- ***uint32_t RTC_TamperTypeDef::PrechargeDuration***
  Specifies the Precharge Duration . This parameter can be a value of ***RTCEx_Tamper_Pin_Precharge_Duration***
- ***uint32_t RTC_TamperTypeDef::TamperPullUp***
  Specifies the Tamper PullUp . This parameter can be a value of ***RTCEx_Tamper_Pull_UP***
- ***uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***

69.1.2 RTC_ATampInputTypeDef

RTC_ATampInputTypeDef is defined in the stm32u5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t Enable***
- ***uint32_t Interrupt***
- ***uint32_t Output***
- ***uint32_t NoErase***
- ***uint32_t MaskFlag***

Field Documentation

- ***uint32_t RTC_ATampInputTypeDef::Enable***
  Specifies the Tamper input is active. This parameter can be a value of ***RTCEx_ActiveTamper_Enable***

- ***uint32_t RTC_ATampInputTypeDef::Interrupt***
  Specifies the interrupt mode This parameter can be a value of ***RTCEx_ActiveTamper_Interrupt***
- ***uint32_t RTC_ATampInputTypeDef::Output***
  Specifies the TAMP output to be compared with. The same output can be used for several tamper inputs. This parameter can be a value of ***RTCEx_ActiveTamper_Sel***
- ***uint32_t RTC_ATampInputTypeDef::NoErase***
  Specifies the Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***
- ***uint32_t RTC_ATampInputTypeDef::MaskFlag***
  Specifies the Tamper Flag masking. This parameter can be a value of ***RTCEx_Tamper_MaskFlag***

69.1.3 RTC_ActiveTampersTypeDef

RTC_ActiveTampersTypeDef is defined in the stm32u5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t ActiveFilter***
- ***uint32_t ActiveAsyncPrescaler***
- ***uint32_t TimeStampOnTamperDetection***
- ***uint32_t ActiveOutputChangePeriod***
- ***uint32_t Seed***
- ***RTC_ATampInputTypeDef TampInput***

Field Documentation

- ***uint32_t RTC_ActiveTampersTypeDef::ActiveFilter***
  Specifies the Active tamper filter enable. This parameter can be a value of ***RTCEx_ActiveTamper_Filter***
- ***uint32_t RTC_ActiveTampersTypeDef::ActiveAsyncPrescaler***
  Specifies the Active Tamper asynchronous Prescaler clock. This parameter can be a value of ***RTCEx_ActiveTamper_Async_prescaler***
- ***uint32_t RTC_ActiveTampersTypeDef::TimeStampOnTamperDetection***
  Specifies the timeStamp on tamper detection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***
- ***uint32_t RTC_ActiveTampersTypeDef::ActiveOutputChangePeriod***
  Specifies the Active Tamper output change period . This parameter can be a value from 0 to 7.
- ***uint32_t RTC_ActiveTampersTypeDef::Seed[RTC_ATAMP_SEED_NB_UINT32]***
  Specifies the RNG Seed value. This parameter is an array of value from 0 to 0xFFFFFFFF
- ***RTC_ATampInputTypeDef RTC_ActiveTampersTypeDef::TampInput[RTC_TAMP_NB]***
  Specifies configuration of all active tampers. The index of TampInput[RTC_TAMP_NB] can be a value of RTCEx_ActiveTamper_Sel

69.1.4 RTC_InternalTamperTypeDef

RTC_InternalTamperTypeDef is defined in the stm32u5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t IntTamper***
- ***uint32_t TimeStampOnTamperDetection***
- ***uint32_t NoErase***

Field Documentation

- ***uint32_t RTC_InternalTamperTypeDef::IntTamper***
  Specifies the Internal Tamper Pin. This parameter can be a value of ***RTCEx_Internal_Tamper_Pins***
- ***uint32_t RTC_InternalTamperTypeDef::TimeStampOnTamperDetection***
  Specifies the TimeStampOnTamperDetection. This parameter can be a value of ***RTCEx_Tamper_TimeStampOnTamperDetection***
- ***uint32_t RTC_InternalTamperTypeDef::NoErase***
  Specifies the internal Tamper no erase mode. This parameter can be a value of ***RTCEx_Tamper_EraseBackUp***

69.1.5 RTC_SecureStateTypeDef

RTC_SecureStateTypeDef is defined in the stm32u5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t rtcSecureFull***
- ***uint32_t rtcNonSecureFeatures***
- ***uint32_t tampSecureFull***
- ***uint32_t backupRegisterStartZone2***
- ***uint32_t backupRegisterStartZone3***
- ***uint32_t MonotonicCounterSecure***

Field Documentation

- ***uint32_t RTC_SecureStateTypeDef::rtcSecureFull***
  Specifies If the RTC is fully secure or not This parameter can be a value of ***RTCEx_RTC_Secure_Full***
- ***uint32_t RTC_SecureStateTypeDef::rtcNonSecureFeatures***
  Specifies the non-secure features. This parameter is only relevant if RTC is not fully secure (rtcSecureFull == RTC_SECURE_FULL_NO). This parameter can be a combination of ***RTCEx_RTC_NonSecure_Features***.
- ***uint32_t RTC_SecureStateTypeDef::tampSecureFull***
  Specifies If the TAMP is fully secure or not execpt monotonic counters and BackUp registers. This parameter can be a value of ***RTCEx_TAMP_Secure_Full***
- ***uint32_t RTC_SecureStateTypeDef::backupRegisterStartZone2***
  Specifies the backup register start zone 2 Zone 1 : read secure write secure. Zone 2 : read non-secure write secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is shared with ***Section 68.3.1: RTC_PrivilegeStateTypeDef***.
- ***uint32_t RTC_SecureStateTypeDef::backupRegisterStartZone3***
  Specifies the backup register start zone 3 Zone 3 : read non-secure write non-secure This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is shared with ***Section 68.3.1: RTC_PrivilegeStateTypeDef***.
- ***uint32_t RTC_SecureStateTypeDef::MonotonicCounterSecure***
  Specifies If the monotonic counter is secure or not This parameter can be a value of ***RTCEx_TAMP_Monotonic_Counter_Secure***

69.1.6 RTC_PrivilegeStateTypeDef

RTC_PrivilegeStateTypeDef is defined in the stm32u5xx_hal_rtc_ex.h

Data Fields

- ***uint32_t rtcPrivilegeFull***
- ***uint32_t rtcPrivilegeFeatures***
- ***uint32_t tampPrivilegeFull***
- ***uint32_t backupRegisterPrivZone***
- ***uint32_t backupRegisterStartZone2***
- ***uint32_t backupRegisterStartZone3***
- ***uint32_t MonotonicCounterPrivilege***

Field Documentation

- ***uint32_t RTC_PrivilegeStateTypeDef::rtcPrivilegeFull***
  Specifies If the RTC is fully privileged or not. This parameter can be a value of ***RTCEx_RTC_Privilege_Full***.
- ***uint32_t RTC_PrivilegeStateTypeDef::rtcPrivilegeFeatures***
  Specifies the privileged features. This parameter is only relevant if RTC is not fully privileged (rtcPrivilegeFull == RTC_PRIVILEGE_FULL_NO). This parameter can be a combination of ***RTCEx_RTC_Privilege_Features***.
- ***uint32_t RTC_PrivilegeStateTypeDef::tampPrivilegeFull***
  Specifies If the TAMP is fully privileged or not execpt monotonic counters and BackUp registers. This parameter can be a value of ***RTCEx_TAMP_Privilege_Full***.
- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterPrivZone***
  Specifies backup register zone to be privileged. This parameter can be a combination of ***RTCEx_Backup_Reg_Privilege_zone***. Warning : this parameter is writable in secure mode or if trustzone is disabled.

- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterStartZone2***
  Specifies the backup register start zone 2. Zone 1 granted accesses : read secure, write secure. Zone 2 granted accesses : read non-secure, write secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register . Warning : this parameter is writable in secure mode or if trustzone is disabled. Warning : this parameter is shared with ***Section 68.3.1: RTC_SecureStateTypeDef***
- ***uint32_t RTC_PrivilegeStateTypeDef::backupRegisterStartZone3***
  Specifies the backup register start zone 3. Zone 3 granted accesses : read non-secure, write non-secure. This parameter can be RTC_BKP_DRx where x can be from 0 to 31 to specify the register. Warning : this parameter is writable in secure mode or if trustzone is disabled. Warning : this parameter is shared with ***Section 68.3.1: RTC_SecureStateTypeDef***.
- ***uint32_t RTC_PrivilegeStateTypeDef::MonotonicCounterPrivilege***
  Specifies If the monotonic counter is privileged or not This parameter can be a value of ***RTCEx_TAMP_Monotonic_Counter_Privilege***

69.2 RTCEx Firmware driver API description

The following section lists the various functions of the RTCEx library.

69.2.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A, Alarm B and WaKeUp signals. To output the selected RTC signal, use the HAL_RTC_Init() function.
- RTC_CALIB: this output is 512Hz signal or 1Hz. To enable the RTC_CALIB, use the HAL_RTCEx_SetCalibrationOutPut() function.
- Two pins can be used as RTC_ALARM or RTC_CALIB (PC13, PB2) managed on the RTC_OR register.
- When the RTC_CALIB or RTC_ALARM output is selected, the RTC_OUT pin is automatically configured in output alternate function.

Smooth digital Calibration configuration

- Configure the RTC Original Digital Calibration Value and the corresponding calibration cycle period (32s,16s and 8s) using the HAL_RTCEx_SetSmoothCalib() function.

TimeStamp configuration

- Enable the RTC TimeStamp using the HAL_RTCEx_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEx_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Internal TimeStamp configuration

- Enable the RTC internal TimeStamp using the HAL_RTCEx_SetInternalTimeStamp() function. User has to check internal timestamp occurrence using __HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEx_GetTimeStamp() function.

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP using the HAL_RTCEx_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the RTC_TAMPCR register.
- With new RTC tamper configuration, you have to call HAL_RTC_Init() in order to perform TAMP base address offset calculation.
- If you do not intend to have tamper using RTC clock, you can bypass its initialization by setting ClockEnable inti field to RTC_CLOCK_DISABLE.
- Enable Internal tamper using HAL_RTCEx_SetInternalTamper. IT mode can be chosen using setting Interrupt field.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEx_BKUPRead() function.
- Before calling these functions you have to call HAL_RTC_Init() in order to perform TAMP base address offset calculation.

69.2.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- *HAL_RTCEx_SetTimeStamp()*
- *HAL_RTCEx_SetTimeStamp_IT()*
- *HAL_RTCEx_DeactivateTimeStamp()*
- *HAL_RTCEx_SetInternalTimeStamp()*
- *HAL_RTCEx_DeactivateInternalTimeStamp()*
- *HAL_RTCEx_GetTimeStamp()*
- *HAL_RTCEx_TimeStampIRQHandler()*
- *HAL_RTCEx_TimeStampEventCallback()*
- *HAL_RTCEx_PollForTimeStampEvent()*

69.2.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

69.2.4 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters

- Set the Smooth calibration parameters.
- Set Low Power calibration parameter.
- Configure the Synchronization Shift Control Settings.
- Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enable the RTC reference clock detection.
- Disable the RTC reference clock detection.
- Enable the Bypass Shadow feature.
- Disable the Bypass Shadow feature.

This section contains the following APIs:
- *HAL_RTCEx_SetSmoothCalib()*
- *HAL_RTCEx_SetLowPowerCalib()*
- *HAL_RTCEx_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*
- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*
- *HAL_RTCEx_MonotonicCounterIncrement()*
- *HAL_RTCEx_MonotonicCounterGet()*
- *HAL_RTCEx_SetSSRU_IT()*
- *HAL_RTCEx_DeactivateSSRU()*
- *HAL_RTCEx_SSRUIRQHandler()*
- *HAL_RTCEx_SSRUEventCallback()*

69.2.5 Extended features functions

This section provides functions allowing to:
- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:
- *HAL_RTCEx_AlarmBEventCallback()*
- *HAL_RTCEx_PollForAlarmBEvent()*

69.2.6 Tamper functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that ine you can select to output tamper event on RTC pin.
- Enable the Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, NoErase, MaskFlag, precharge or discharge and Pull-UP, timestamp using the HAL_RTCEx_SetTamper() function. You can configure Tamper with interrupt mode using HAL_RTCEx_SetTamper_IT() function.
- The default configuration of the Tamper erases the backup registers. To avoid erase, enable the NoErase field on the TAMP_TAMPCR register.
- Enable Internal Tamper and configure it with interrupt, timestamp using the HAL_RTCEx_SetInternalTamper() function.

This section contains the following APIs:
- *HAL_RTCEx_SetTamper()*
- *HAL_RTCEx_SetTamper_IT()*
- *HAL_RTCEx_DeactivateTamper()*
- *HAL_RTCEx_SetActiveTampers()*

- *HAL_RTCEx_GetActiveTampers()*
- *HAL_RTCEx_SetActiveSeed()*
- *HAL_RTCEx_SetBoothardwareKey()*
- *HAL_RTCEx_DeactivateActiveTampers()*
- *HAL_RTCEx_PollForTamperEvent()*
- *HAL_RTCEx_SetInternalTamper()*
- *HAL_RTCEx_GetInternalTampers()*
- *HAL_RTCEx_SetInternalTamper_IT()*
- *HAL_RTCEx_DeactivateInternalTamper()*
- *HAL_RTCEx_PollForInternalTamperEvent()*
- *HAL_RTCEx_TamperIRQHandler()*
- *HAL_RTCEx_Tamper1EventCallback()*
- *HAL_RTCEx_Tamper2EventCallback()*
- *HAL_RTCEx_Tamper3EventCallback()*
- *HAL_RTCEx_Tamper4EventCallback()*
- *HAL_RTCEx_Tamper5EventCallback()*
- *HAL_RTCEx_Tamper6EventCallback()*
- *HAL_RTCEx_Tamper7EventCallback()*
- *HAL_RTCEx_Tamper8EventCallback()*
- *HAL_RTCEx_InternalTamper1EventCallback()*
- *HAL_RTCEx_InternalTamper2EventCallback()*
- *HAL_RTCEx_InternalTamper3EventCallback()*
- *HAL_RTCEx_InternalTamper5EventCallback()*
- *HAL_RTCEx_InternalTamper6EventCallback()*
- *HAL_RTCEx_InternalTamper7EventCallback()*
- *HAL_RTCEx_InternalTamper8EventCallback()*
- *HAL_RTCEx_InternalTamper9EventCallback()*
- *HAL_RTCEx_InternalTamper11EventCallback()*
- *HAL_RTCEx_InternalTamper12EventCallback()*
- *HAL_RTCEx_InternalTamper13EventCallback()*

69.2.7 Extended RTC Backup register functions

- Before calling any tamper or internal tamper function, you have to call first HAL_RTC_Init() function.
- In that one you can select to output tamper event on RTC pin.

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register

This section contains the following APIs:

- *HAL_RTCEx_BKUPWrite()*
- *HAL_RTCEx_BKUPRead()*
- *HAL_RTCEx_BKUPErase()*
- *HAL_RTCEx_BKUPBlock_Enable()*
- *HAL_RTCEx_BKUPBlock_Disable()*
- *HAL_RTCEx_Erase_SecretDev_Conf()*
- *HAL_RTCEx_Get_Erase_SecretDev_Conf()*

69.2.8 Extended RTC security functions

- Before calling security function, you have to call first HAL_RTC_Init() function.

This section contains the following APIs:

- *HAL_RTCEx_SecureModeGet()*
- *HAL_RTCEx_SecureModeSet()*

69.2.9 Extended RTC privilege functions

- Before calling privilege function, you have to call first HAL_RTC_Init() function.

This section contains the following APIs:

- *HAL_RTCEx_PrivilegeModeSet()*
- *HAL_RTCEx_PrivilegeModeGet()*

69.2.10 Detailed description of functions

HAL_RTCEx_SetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  – RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  – RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  – RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_SetTimeStamp_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge, uint32_t RTC_TimeStampPin)

Function description

Set TimeStamp with Interrupt.

Parameters

- **hrtc:** RTC handle
- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values:
  - RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin.
  - RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
- **RTC_TimeStampPin:** Specifies the RTC TimeStamp Pin. This parameter can be one of the following values:
  - RTC_TIMESTAMPPIN_DEFAULT: PC13 is selected as RTC TimeStamp Pin. The RTC TimeStamp Pin is per default PC13, but for reasons of compatibility, this parameter is required.

Return values

- **HAL:** status

Notes

- This API must be called before enabling the TimeStamp feature.

HAL_RTCEx_DeactivateTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Set Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- This API must be called before enabling the internal TimeStamp feature.

HAL_RTCEx_DeactivateInternalTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTimeStamp (RTC_HandleTypeDef * hrtc)

Function description

Deactivate Internal TimeStamp.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetTimeStamp

Function name

HAL_StatusTypeDef HAL_RTCEx_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function description

Get the RTC TimeStamp value.

Parameters

- **hrtc:** RTC handle
- **sTimeStamp:** Pointer to Time structure
- **sTimeStampDate:** Pointer to Date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
  – RTC_FORMAT_BIN: Binary data format
  – RTC_FORMAT_BCD: BCD data format

Return values

- **HAL:** status

HAL_RTCEx_TimeStampIRQHandler

Function name

void HAL_RTCEx_TimeStampIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle TimeStamp interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForTimeStampEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle TimeStamp polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_TimeStampEventCallback

Function name

void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)

Function description

TimeStamp callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Set wake up timer.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock, uint32_t WakeUpAutoClr)

Function description

Set wake up timer with interrupt.

Parameters

- **hrtc:** RTC handle
- **WakeUpCounter:** Wake up counter
- **WakeUpClock:** Wake up clock
- **WakeUpAutoClr:** Wake up auto clear value (look at WUTOCLR in reference manual)
  – No effect if WakeUpAutoClr is set to zero
  – This feature is meaningful in case of Low power mode to avoid any RTC software execution after Wake Up.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Deactivate wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Get wake up timer counter.

Parameters

- **hrtc:** RTC handle

Return values

- **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Wake Up Timer interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name

void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Wake Up Timer callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Wake Up Timer Polling.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetSmoothCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)

Function description

Set the Smooth calibration parameters.

Parameters

- **hrtc:** RTC handle
- **SmoothCalibPeriod:** Select the Smooth Calibration Period. This parameter can be one of the following values :
  - RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s.
  - RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s.
  - RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s.
- **SmoothCalibPlusPulses:** Select to Set or reset the CALP bit. This parameter can be one of the following values:
  - RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK pulse every 2*11 pulses.
  - RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added.
- **SmoothCalibMinusPulsesValue:** Select the value of CALM[8:0] bits. This parameter can be one any value from 0 to 0x000001FF.

Return values

- **HAL:** status

Notes

- To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

HAL_RTCEx_SetLowPowerCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetLowPowerCalib (RTC_HandleTypeDef * hrtc, uint32_t LowPowerCalib)

Function description

Select the low power Calibration mode.

Parameters

- **hrtc:** RTC handle
- **LowPowerCalib:** Low power Calibration mode. This parameter can be can be one of the following values :
  - RTC_LPCAL_SET: Low power mode.
  - RTC_LPCAL_RESET: High consumption mode.

Return values

- **HAL:** status

HAL_RTCEx_SetSynchroShift

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)

Function description

Configure the Synchronization Shift Control Settings.

Parameters

- **hrtc:** RTC handle
- **ShiftAdd1S:** Select to add or not 1 second to the time calendar. This parameter can be one of the following values:
  – RTC_SHIFTADD1S_SET: Add one second to the clock calendar.
  – RTC_SHIFTADD1S_RESET: No effect.
- **ShiftSubFS:** Select the number of Second Fractions to substitute. This parameter can be one any value from 0 to 0x7FFF.

Return values

- **HAL:** status

Notes

- When REFCKON is set, firmware must not write to Shift control register.

HAL_RTCEx_SetCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function description

Configure the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle
- **CalibOutput:** Select the Calibration output Selection . This parameter can be one of the following values:
  – RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.
  – RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)

Function description

Deactivate the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SetRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)

Function description

Enable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

Disable the RTC reference clock detection.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Enable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_DisableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_DisableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Disable the Bypass Shadow feature.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

HAL_RTCEx_MonotonicCounterIncrement

Function name

HAL_StatusTypeDef HAL_RTCEx_MonotonicCounterIncrement (RTC_HandleTypeDef * hrtc, uint32_t Instance)

Function description

Increment Monotonic counter.

Parameters

- **hrtc:** RTC handle
- **Instance:** Monotonic counter Instance This parameter can be can be one of the following values :
  – RTC_MONOTONIC_COUNTER_1

Return values

- **HAL:** status

HAL_RTCEx_MonotonicCounterGet

Function name

HAL_StatusTypeDef HAL_RTCEx_MonotonicCounterGet (RTC_HandleTypeDef * hrtc, uint32_t Instance, uint32_t * Value)

Function description

Monotonic counter incrementation.

Parameters

- **hrtc:** RTC handle
- **Instance:** Monotonic counter Instance This parameter can be can be one of the following values :
  – RTC_MONOTONIC_COUNTER_1
- **Value:** Pointer to the counter monotonic counter value

Return values

- **HAL:** status

HAL_RTCEx_SetSSRU_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSSRU_IT (RTC_HandleTypeDef * hrtc)

Function description

Set SSR Underflow detection with Interrupt.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_DeactivateSSRU

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateSSRU (RTC_HandleTypeDef * hrtc)

Function description

Deactivate SSR Underflow.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_SSRUIRQHandler

Function name

void HAL_RTCEx_SSRUIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle SSR underflow interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_SSRUEventCallback

Function name

void HAL_RTCEx_SSRUEventCallback (RTC_HandleTypeDef * hrtc)

Function description

SSR underflow callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_AlarmBEventCallback

Function name

void HAL_RTCEx_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Alarm B callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_PollForAlarmBEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForAlarmBEvent (const RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handle Alarm B Polling request.

Parameters

- **hrtc:** RTC handle
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function description

Set Tamper.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetActiveTampers

Function name

HAL_StatusTypeDef HAL_RTCEx_SetActiveTampers (RTC_HandleTypeDef * hrtc, RTC_ActiveTampersTypeDef * sAllTamper)

Function description

Set all active Tampers at the same time.

Parameters

- **hrtc:** RTC handle
- **sAllTamper:** Pointer to active Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetActiveSeed

Function name

HAL_StatusTypeDef HAL_RTCEx_SetActiveSeed (RTC_HandleTypeDef * hrtc, uint32_t * pSeed)

Function description

Write a new seed.

Parameters

- **hrtc:** RTC handle
- **pSeed:** Pointer to active tamper seed values.

Return values

- **HAL:** status

HAL_RTCEx_SetTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function description

Set Tamper in IT mode.

Parameters

- **hrtc:** RTC handle
- **sTamper:** Pointer to Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)

Function description

Deactivate Tamper.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be a combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2
  - RTC_TAMPER_3
  - RTC_TAMPER_4
  - RTC_TAMPER_5
  - RTC_TAMPER_6
  - RTC_TAMPER_7
  - RTC_TAMPER_8

Return values

- **HAL:** status

HAL_RTCEx_DeactivateActiveTampers

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateActiveTampers (RTC_HandleTypeDef * hrtc)

Function description

Deactivate all Active Tampers at the same time.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

HAL_RTCEx_PollForTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForTamperEvent (RTC_HandleTypeDef * hrtc, uint32_t Tamper, uint32_t Timeout)

Function description

Tamper event polling.

Parameters

- **hrtc:** RTC handle
- **Tamper:** Selected tamper pin. This parameter can be a combination of the following values:
  - RTC_TAMPER_1
  - RTC_TAMPER_2
  - RTC_TAMPER_3
  - RTC_TAMPER_4
  - RTC_TAMPER_5
  - RTC_TAMPER_6
  - RTC_TAMPER_7
  - RTC_TAMPER_8
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper (RTC_HandleTypeDef * hrtc, RTC_InternalTamperTypeDef * sIntTamper)

Function description

Set Internal Tamper.

Parameters

- **hrtc:** RTC handle
- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_SetInternalTamper_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetInternalTamper_IT (RTC_HandleTypeDef * hrtc, RTC_InternalTamperTypeDef * sIntTamper)

Function description

Set Internal Tamper in interrupt mode.

Parameters

- **hrtc:** RTC handle
- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_DeactivateInternalTamper

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateInternalTamper (RTC_HandleTypeDef * hrtc, uint32_t IntTamper)

Function description

Deactivate Internal Tamper.

Parameters

- **hrtc:** RTC handle
- **IntTamper:** Selected internal tamper event. This parameter can be any combination of existing internal tampers.

Return values

- **HAL:** status

HAL_RTCEx_PollForInternalTamperEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForInternalTamperEvent (RTC_HandleTypeDef * hrtc, uint32_t IntTamper, uint32_t Timeout)

Function description

Internal Tamper event polling.

Parameters

- **hrtc:** RTC handle
- **IntTamper:** selected tamper. This parameter can be any combination of existing internal tampers.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_SetBoothardwareKey

Function name

HAL_StatusTypeDef HAL_RTCEx_SetBoothardwareKey (RTC_HandleTypeDef * hrtc)

Function description

Lock the Boot hardware Key.

Parameters

- **hrtc:** RTC handle

Return values

- **HAL:** status

Notes

- The backup registers from TAMP_BKP0R to TAMP_BKP7R cannot be accessed neither in read nor in write (they are read as 0 and write ignore).

HAL_RTCEx_GetActiveTampers

Function name

void HAL_RTCEx_GetActiveTampers (RTC_ActiveTampersTypeDef * sAllTamper)

Function description

Get active Tampers configuration.

Parameters

- **sAllTamper:** Pointer to active Tamper Structure.

Return values

- **none:**

HAL_RTCEx_GetInternalTampers

Function name

void HAL_RTCEx_GetInternalTampers (RTC_InternalTamperTypeDef * sIntTamper)

Function description

Get Internal Tamper Configuration.

Parameters

- **sIntTamper:** Pointer to Internal Tamper Structure.

Return values

- **HAL:** status

HAL_RTCEx_TamperIRQHandler

Function name

void HAL_RTCEx_TamperIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Tamper interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper1EventCallback

Function name

void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper2EventCallback

Function name

void HAL_RTCEx_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper3EventCallback

Function name

void HAL_RTCEx_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper4EventCallback

Function name

void HAL_RTCEx_Tamper4EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 4 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper5EventCallback

Function name

void HAL_RTCEx_Tamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 5 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper6EventCallback

Function name

void HAL_RTCEx_Tamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 6 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper7EventCallback

Function name

void HAL_RTCEx_Tamper7EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 7 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Tamper8EventCallback

Function name

void HAL_RTCEx_Tamper8EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Tamper 8 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper1EventCallback

Function name

void HAL_RTCEx_InternalTamper1EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 1 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper2EventCallback

Function name

void HAL_RTCEx_InternalTamper2EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 2 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper3EventCallback

Function name

void HAL_RTCEx_InternalTamper3EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 3 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper5EventCallback

Function name

void HAL_RTCEx_InternalTamper5EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 5 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper6EventCallback

Function name

void HAL_RTCEx_InternalTamper6EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 6 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper7EventCallback

Function name

void HAL_RTCEx_InternalTamper7EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 7 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper8EventCallback

Function name

void HAL_RTCEx_InternalTamper8EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 8 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper9EventCallback

Function name

void HAL_RTCEx_InternalTamper9EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 9 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper11EventCallback

Function name

void HAL_RTCEx_InternalTamper11EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 11 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper12EventCallback

Function name

void HAL_RTCEx_InternalTamper12EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 12 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_InternalTamper13EventCallback

Function name

void HAL_RTCEx_InternalTamper13EventCallback (RTC_HandleTypeDef * hrtc)

Function description

Internal Tamper 13 callback.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPWrite

Function name

void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)

Function description

Write a data in a specified RTC Backup data register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be RTC_BKP_DRx where x can be from 0 to RTC_BACKUP_NB
- **Data:** Data to be written in the specified Backup data register.

Return values

- **None:**

HAL_RTCEx_BKUPRead

Function name

uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **hrtc:** RTC handle
- **BackupRegister:** RTC Backup data Register number. This parameter can be RTC_BKP_DRx where x can be from 0 to RTC_BACKUP_NB

Return values

- **Read:** value

HAL_RTCEx_BKUPErase

Function name

void HAL_RTCEx_BKUPErase (RTC_HandleTypeDef * hrtc)

Function description

Reset the RTC Backup data Register and the device secrets.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPBlock_Enable

Function name

void HAL_RTCEx_BKUPBlock_Enable (RTC_HandleTypeDef * hrtc)

Function description

Block the access to the RTC Backup data Register and the device secrets.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_BKUPBlock_Disable

Function name

void HAL_RTCEx_BKUPBlock_Disable (RTC_HandleTypeDef * hrtc)

Function description

Disable the Block to the access to the RTC Backup data Register and the device secrets.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTCEx_Erase_SecretDev_Conf

Function name

void HAL_RTCEx_Erase_SecretDev_Conf (RTC_HandleTypeDef * hrtc, uint32_t SecretDeviceConf)

Function description

Enable and Disable the erase of the configurable Device Secerts.

Parameters

- **hrtc:** RTC handle
- **SecretDeviceConf:** Specifies the configuration of the Secrets Devices This parameter can be one of the following values:
  – TAMP_SECRETDEVICE_ERASE_ENABLE: Configurable device secrets are is included in the device secrets protected by TAMP peripheral.
  – TAMP_SECRETDEVICE_ERASE_DISABLE: Configurable device secrets are not included in the device secrets protected by TAMP peripheral.

Return values

- **None:**

Notes

- This API must be called before enabling the Tamper.

HAL_RTCEx_Get_Erase_SecretDev_Conf

Function name

uint32_t HAL_RTCEx_Get_Erase_SecretDev_Conf (void)

Function description

Get the erase configuration of the Device Secerts.

Return values

- **RTCEx_TAMP_Secret_Device_Conf_Erase:**

HAL_RTCEx_SecureModeGet

Function name

HAL_StatusTypeDef HAL_RTCEx_SecureModeGet (RTC_HandleTypeDef * hrtc, RTC_SecureStateTypeDef * secureState)

Function description

Get the security level of the RTC.

Parameters

- **hrtc:** RTC handle
- **secureState:** Secure state

Return values

- **HAL_StatusTypeDef:**

HAL_RTCEx_SecureModeSet

Function name

HAL_StatusTypeDef HAL_RTCEx_SecureModeSet (RTC_HandleTypeDef * hrtc, RTC_SecureStateTypeDef * secureState)

Function description

Set the security level of the RTC/TAMP/Backup registers.

Parameters

- **hrtc:** RTC handle
- **secureState:** Secure state

Return values

- **HAL_StatusTypeDef:**

HAL_RTCEx_PrivilegeModeSet

Function name

HAL_StatusTypeDef HAL_RTCEx_PrivilegeModeSet (RTC_HandleTypeDef * hrtc, RTC_PrivilegeStateTypeDef * privilegeState)

Function description

Set the privilege level of the RTC/TAMP registers.

Parameters

- **hrtc:** RTC handle
- **privilegeState:** Privilege state

Return values

- **HAL_StatusTypeDef:**

HAL_RTCEx_PrivilegeModeGet

Function name

HAL_StatusTypeDef HAL_RTCEx_PrivilegeModeGet (RTC_HandleTypeDef * hrtc, RTC_PrivilegeStateTypeDef * privilegeState)

Function description

Get the privilege level of the RTC.

Parameters

- **hrtc:** RTC handle
- **privilegeState:** Privilege state

Return values

- **HAL_StatusTypeDef:**

69.3 RTCEx Firmware driver defines

The following section lists the various define and macros of the module.

69.3.1 RTCEx

RTCEx

RTCEx Active_Tamper_Asynchronous_Prescaler clock Definitions

RTC_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

RTC_ATAMP_ASYNCPRES_RTCCLK_2

RTCCLK/2

RTC_ATAMP_ASYNCPRES_RTCCLK_4

RTCCLK/4

RTC_ATAMP_ASYNCPRES_RTCCLK_8

RTCCLK/8

RTC_ATAMP_ASYNCPRES_RTCCLK_16

RTCCLK/16

RTC_ATAMP_ASYNCPRES_RTCCLK_32

RTCCLK/32

RTC_ATAMP_ASYNCPRES_RTCCLK_64

RTCCLK/64

RTC_ATAMP_ASYNCPRES_RTCCLK_128

RTCCLK/128

RTC_ATAMP_ASYNCPRES_RTCCLK_2048

RTCCLK/2048

RTCEx_ActiveTamper_Enable Definitions

RTC_ATAMP_ENABLE

RTC_ATAMP_DISABLE

RTCEx_ActiveTamper_Filter Definitions

RTC_ATAMP_FILTER_ENABLE

RTC_ATAMP_FILTER_DISABLE

RTCEx_ActiveTamper_Interrupt Definitions

RTC_ATAMP_INTERRUPT_ENABLE

RTC_ATAMP_INTERRUPT_DISABLE

RTCEx Active Tamper selection Definition

RTC_ATAMP_1

Tamper 1

RTC_ATAMP_2

Tamper 2

RTC_ATAMP_3

Tamper 3

RTC_ATAMP_4

Tamper 4

RTC_ATAMP_5

Tamper 5

RTC_ATAMP_6

Tamper 6

RTC_ATAMP_7

Tamper 7

RTC_ATAMP_8

Tamper 8

Seed size Definitions

RTC_ATAMP_SEED_NB_UINT32

RTCEx Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

RTC Alarm Sub Seconds

RTC_ALARMSUBSECONDBIN_AUTOCLR_NO

The synchronous Binary counter (SS[31:0] in RTC_SSR) is free-running.

RTC_ALARMSUBSECONDBIN_AUTOCLR_YES

The synchronous Binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF whenreaching RTC_ALRMABINR -> SS[31:0].

RTC Alarm Sub Seconds with binary mode Masks Definitions

RTC_ALARMSUBSECONDBINMASK_ALL

All Alarm SS fields are masked.There is no comparison on sub seconds for Alarm

RTC_ALARMSUBSECONDBINMASK_SS31_1

SS[31:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDBINMASK_SS31_2

SS[31:2] are don't care in Alarm comparison. Only SS[1:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_3

SS[31:3] are don't care in Alarm comparison. Only SS[2:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_4

SS[31:4] are don't care in Alarm comparison. Only SS[3:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_5

SS[31:5] are don't care in Alarm comparison. Only SS[4:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_6

SS[31:6] are don't care in Alarm comparison. Only SS[5:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_7

SS[31:7] are don't care in Alarm comparison. Only SS[6:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_8

SS[31:8] are don't care in Alarm comparison. Only SS[7:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_9

SS[31:9] are don't care in Alarm comparison. Only SS[8:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_10

SS[31:10] are don't care in Alarm comparison. Only SS[9:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_11

SS[31:11] are don't care in Alarm comparison. Only SS[10:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_12

SS[31:12] are don't care in Alarm comparison.Only SS[11:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_13

SS[31:13] are don't care in Alarm comparison. Only SS[12:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_14
SS[31:14] are don't care in Alarm comparison. Only SS[13:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_15
SS[31:15] are don't care in Alarm comparison. Only SS[14:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_16
SS[31:16] are don't care in Alarm comparison. Only SS[15:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_17
SS[31:17] are don't care in Alarm comparison. Only SS[16:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_18
SS[31:18] are don't care in Alarm comparison. Only SS[17:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_19
SS[31:19] are don't care in Alarm comparison. Only SS[18:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_20
SS[31:20] are don't care in Alarm comparison. Only SS[19:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_21
SS[31:21] are don't care in Alarm comparison. Only SS[20:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_22
SS[31:22] are don't care in Alarm comparison. Only SS[21:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_23
SS[31:23] are don't care in Alarm comparison. Only SS[22:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_24
SS[31:24] are don't care in Alarm comparison. Only SS[23:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_25
SS[31:25] are don't care in Alarm comparison. Only SS[24:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_26
SS[31:26] are don't care in Alarm comparison. Only SS[25:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_27
SS[31:27] are don't care in Alarm comparison. Only SS[26:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_28
SS[31:28] are don't care in Alarm comparison. Only SS[27:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_29
SS[31:29] are don't care in Alarm comparison. Only SS[28:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31_30
SS[31:30] are don't care in Alarm comparison. Only SS[29:0] are compared

RTC_ALARMSUBSECONDBINMASK_SS31
SS[31] is don't care in Alarm comparison. Only SS[30:0] are compared

RTC_ALARMSUBSECONDBINMASK_NONE
SS[31:0] are compared and must match to activate alarm.

RTCEx Backup Registers Definition

RTC_BKP_NUMBER

RTC_BKP_DR0

RTC_BKP_DR1

RTC_BKP_DR2

RTC_BKP_DR3

RTC_BKP_DR4

RTC_BKP_DR5

RTC_BKP_DR6

RTC_BKP_DR7

RTC_BKP_DR8

RTC_BKP_DR9

RTC_BKP_DR10

RTC_BKP_DR11

RTC_BKP_DR12

RTC_BKP_DR13

RTC_BKP_DR14

RTC_BKP_DR15

RTC_BKP_DR16

RTC_BKP_DR17

RTC_BKP_DR18

RTC_BKP_DR19

RTC_BKP_DR20

RTC_BKP_DR21

RTC_BKP_DR22

RTC_BKP_DR23

RTC_BKP_DR24

RTC_BKP_DR25

RTC_BKP_DR26

RTC_BKP_DR27

RTC_BKP_DR28

RTC_BKP_DR29

RTC_BKP_DR30

RTC_BKP_DR31

RTCEx Privilege Backup register privilege zone Definition

RTC_PRIVILEGE_BKUP_ZONE_NONE

RTC_PRIVILEGE_BKUP_ZONE_1

RTC_PRIVILEGE_BKUP_ZONE_2

RTC_PRIVILEGE_BKUP_ZONE_ALL

If Binary mode is RTC_BINARY_MIX, the BCD calendar second is incremented

RTC_BINARY_MIX_BCDU_0

The 1s BCD calendar increment is generated each time SS[7:0] = 0

RTC_BINARY_MIX_BCDU_1

The 1s BCD calendar increment is generated each time SS[8:0] = 0

RTC_BINARY_MIX_BCDU_2

The 1s BCD calendar increment is generated each time SS[9:0] = 0

RTC_BINARY_MIX_BCDU_3

The 1s BCD calendar increment is generated each time SS[10:0] = 0

RTC_BINARY_MIX_BCDU_4

The 1s BCD calendar increment is generated each time SS[11:0] = 0

RTC_BINARY_MIX_BCDU_5

The 1s BCD calendar increment is generated each time SS[12:0] = 0

RTC_BINARY_MIX_BCDU_6

The 1s BCD calendar increment is generated each time SS[13:0] = 0

RTC_BINARY_MIX_BCDU_7

The 1s BCD calendar increment is generated each time SS[14:0] = 0

RTC Binary Mode (32-bit free-running counter configuration).

RTC_BINARY_NONE

Free running BCD calendar mode (Binary mode disabled)

RTC_BINARY_ONLY

Free running Binary mode (BCD mode disabled)

RTC_BINARY_MIX

Free running BCD calendar and Binary modes

RTC Calibration

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Description:
- Enable the RTC calibration output.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Description:
- Disable the calibration output.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_CLOCKREF_DETECTION_ENABLE

Description:
- Enable the clock reference detection.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_CLOCKREF_DETECTION_DISABLE

Description:
- Disable the clock reference detection.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_SHIFT_GET_FLAG

Description:
- Get the selected RTC shift operations flag status.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC shift operation Flag is pending or not. This parameter can be:
  – RTC_FLAG_SHPF

Return value:
- The: state of __FLAG__ (TRUE or FALSE)

RTCEx Calib Output selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

RTCEx Exported Macros

__HAL_RTC_CLEAR_FLAG

Description:
- Clear the specified RTC pending flag.

Parameters:
- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_CLEAR_ITSF Clear Internal Time-stamp flag
  - RTC_CLEAR_TSOVF Clear Time-stamp overflow flag
  - RTC_CLEAR_TSF Clear Time-stamp flag
  - RTC_CLEAR_WUTF Clear Wakeup timer flag
  - RTC_CLEAR_ALRBF Clear Alarm B flag
  - RTC_CLEAR_ALRAF Clear Alarm A flag

Return value:
- None

__HAL_RTC_GET_FLAG

Description:
- Check whether the specified RTC flag is set or not.

Parameters:
- __HANDLE__: specifies the RTC Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - RTC_FLAG_RECALPF Recalibration pending Flag
  - RTC_FLAG_INITF Initialization flag
  - RTC_FLAG_RSF Registers synchronization flag
  - RTC_FLAG_INITS Initialization status flag
  - RTC_FLAG_SHPF Shift operation pending flag
  - RTC_FLAG_WUTWF Wakeup timer write flag
  - RTC_FLAG_ITSF Internal Time-stamp flag
  - RTC_FLAG_TSOVF Time-stamp overflow flag
  - RTC_FLAG_TSF Time-stamp flag
  - RTC_FLAG_WUTF Wakeup timer flag
  - RTC_FLAG_ALRBF Alarm B flag
  - RTC_FLAG_ALRAF Alarm A flag

Return value:
- The: state of __FLAG__ (TRUE or FALSE).

RTCEx Flags

RTC_FLAG_TAMP_1

RTC_FLAG_TAMP_2

RTC_FLAG_TAMP_3

RTC_FLAG_TAMP_4

RTC_FLAG_TAMP_5

RTC_FLAG_TAMP_6

RTC_FLAG_TAMP_7

RTC_FLAG_TAMP_8

RTC_FLAG_TAMP_ALL

RTC_FLAG_INT_TAMP_1

RTC_FLAG_INT_TAMP_2

RTC_FLAG_INT_TAMP_3

RTC_FLAG_INT_TAMP_5

RTC_FLAG_INT_TAMP_6

RTC_FLAG_INT_TAMP_7

RTC_FLAG_INT_TAMP_8

RTC_FLAG_INT_TAMP_9

RTC_FLAG_INT_TAMP_11

RTC_FLAG_INT_TAMP_12

RTC_FLAG_INT_TAMP_13

RTC_FLAG_INT_TAMP_ALL

RTCEx Internal Tamper Interrupt

RTC_IT_TAMP_1

Tamper 1 Interrupt

RTC_IT_TAMP_2

Tamper 2 Interrupt

RTC_IT_TAMP_3

Tamper 3 Interrupt

RTC_IT_TAMP_4

Tamper 4 Interrupt

RTC_IT_TAMP_5

Tamper 5 Interrupt

RTC_IT_TAMP_6

Tamper 6 Interrupt

RTC_IT_TAMP_7

Tamper 7 Interrupt

RTC_IT_TAMP_8

Tamper 8 Interrupt

RTC_IT_TAMP_ALL

RTC_IT_INT_TAMP_1

Tamper 1 internal Interrupt

RTC_IT_INT_TAMP_2

Tamper 2 internal Interrupt

RTC_IT_INT_TAMP_3

Tamper 3 internal Interrupt

RTC_IT_INT_TAMP_5

Tamper 5 internal Interrupt

RTC_IT_INT_TAMP_6

Tamper 6 internal Interrupt

RTC_IT_INT_TAMP_7

Tamper 7 internal Interrupt

RTC_IT_INT_TAMP_8

Tamper 8 internal Interrupt

RTC_IT_INT_TAMP_9

Tamper 9 internal Interrupt

RTC_IT_INT_TAMP_11

Tamper 11 internal Interrupt

RTC_IT_INT_TAMP_12

Tamper 12 internal Interrupt

RTC_IT_INT_TAMP_13

Tamper 13 internal Interrupt

RTC_IT_INT_TAMP_ALL

RTCEx Internal Tamper Pins Definition

RTC_INT_TAMPER_1

RTC_INT_TAMPER_2

RTC_INT_TAMPER_3

RTC_INT_TAMPER_5

RTC_INT_TAMPER_6

RTC_INT_TAMPER_7

RTC_INT_TAMPER_8

RTC_INT_TAMPER_9

RTC_INT_TAMPER_11

RTC_INT_TAMPER_12

RTC_INT_TAMPER_13

RTC_INT_TAMPER_ALL

Private macros to check input parameters

IS_TIMESTAMP_EDGE

IS_RTC_TIMESTAMP_PIN

IS_RTC_TIMESTAMPONTAMPER_DETECTION

IS_RTC_TAMPER_TAMPERDETECTIONOUTPUT

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_LOW_POWER_CALIB

IS_RTC_TAMPER

IS_RTC_INTERNAL_TAMPER

IS_RTC_TAMPER_TRIGGER

IS_RTC_TAMPER_ERASE_MODE

IS_RTC_TAMPER_MASKFLAG_STATE

IS_RTC_TAMPER_FILTER

IS_RTC_TAMPER_SAMPLING_FREQ

IS_RTC_TAMPER_PRECHARGE_DURATION

IS_RTC_TAMPER_PULLUP_STATE

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

IS_RTC_BKP

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

IS_RTC_SECURE_FULL

IS_RTC_NONSECURE_FEATURES

IS_TAMP_SECURE_FULL

IS_TAMP_MONOTONIC_CNT_SECURE

IS_RTC_PRIVILEGE_FULL

IS_RTC_PRIVILEGE_FEATURES

IS_TAMP_PRIVILEGE_FULL

IS_TAMP_MONOTONIC_CNT_PRIVILEGE

IS_RTC_PRIVILEGE_BKUP_ZONE

IS_RTC_BINARY_MODE

IS_RTC_BINARY_MIX_BCDU

IS_RTC_ALARM_SUB_SECOND_BINARY_MASK

IS_RTC_ALARMSUBSECONDBIN_AUTOCLR

RTCEx Monotonic Counter Instance Definition

RTC_MONOTONIC_COUNTER_1

Monotonic counter 1

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_ALARMB

RTC_OUTPUT_WAKEUP

RTC_OUTPUT_TAMPER

RTCEx Secure Features Definition

RTC_NONSECURE_FEATURE_NONE

RTC_NONSECURE_FEATURE_INIT

Initialization

RTC_NONSECURE_FEATURE_CAL

Calibration

RTC_NONSECURE_FEATURE_TS

Time stamp

RTC_NONSECURE_FEATURE_WUT

Wake up timer

RTC_NONSECURE_FEATURE_ALRA

Alarm A

RTC_NONSECURE_FEATURE_ALRB

Alarm B

RTC_NONSECURE_FEATURE_ALL

RTCEx Privilege Features Definition

RTC_PRIVILEGE_FEATURE_NONE

RTC_PRIVILEGE_FEATURE_INIT

Initialization

RTC_PRIVILEGE_FEATURE_CAL

Calibration

RTC_PRIVILEGE_FEATURE_TS

Time stamp

RTC_PRIVILEGE_FEATURE_WUT

Wake up timer

RTC_PRIVILEGE_FEATURE_ALRA

Alarm A

RTC_PRIVILEGE_FEATURE_ALRB

Alarm B

RTC_PRIVILEGE_FEATURE_ALL

RTCEx Privilege Full Definition

RTC_PRIVILEGE_FULL_YES

RTC_PRIVILEGE_FULL_NO

RTCEx Secure Definition

RTC_SECURE_FULL_YES

RTC full secure

RTC_SECURE_FULL_NO

RTC is not full secure, features can be unsecure. See RTC_LL_EC_UNSECURE_RTC_FEATURE

RTCEx Smooth calib Low Power Definitions

RTC_LPCAL_SET

Calibration window is 2exp20 ck_apre, which is the required configuration for ultra-low consumption mode.

RTC_LPCAL_RESET

Calibration window is 2exp20 RTCCLK, which is a high-consumption mode. This mode should be set only when less than 32s calibration window is required.

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK pulses

RTCEx Smooth calib Plus pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET

The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET

The number of RTCCLK pulses subbstited during a 32-second window = CALM[8:0]

RTC SSR Underflow

__HAL_RTC_SSRU_ENABLE_IT

Description:

- Enable the RTC SSRU interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to be enabled. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_DISABLE_IT

Description:

- Disable the RTC SSRU interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to be disabled. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- None

__HAL_RTC_SSRU_GET_IT

Description:

- Check whether the specified RTC SSRU interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt to check. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_SSRU_GET_IT_SOURCE

Description:

- Check whether the specified RTC SSRU interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC SSRU interrupt sources to check. This parameter can be:
  - RTC_IT_SSRU SSRU interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_SSRU_GET_FLAG

Description:

- Get the selected RTC SSRU's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC SSRU Flag is pending or not. This parameter can be:
  - RTC_FLAG_SSRUF

Return value:

- The: state of __FLAG__ (TRUE or FALSE)

__HAL_RTC_SSRU_CLEAR_FLAG

Description:

- Clear the RTC SSRU's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC SSRU Flag to clear. This parameter can be:
  - RTC_FLAG_SSRUF

Return value:

- None

RTCEx tamper

__HAL_RTC_TAMPER_ENABLE

Description:

- Enable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper source to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2
  - RTC_TAMPER_3: Tamper3
  - RTC_TAMPER_4: Tamper4
  - RTC_TAMPER_5: Tamper5
  - RTC_TAMPER_6: Tamper6
  - RTC_TAMPER_7: Tamper7
  - RTC_TAMPER_8: Tamper8

Return value:

- None

__HAL_RTC_TAMPER_DISABLE

Description:

- Disable the TAMP Tamper input detection.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __TAMPER__: specifies the RTC Tamper sources to be enabled. This parameter can be any combination of the following values:
  - RTC_TAMPER_ALL: All tampers
  - RTC_TAMPER_1: Tamper1
  - RTC_TAMPER_2: Tamper2
  - RTC_TAMPER_3: Tamper3
  - RTC_TAMPER_4: Tamper4
  - RTC_TAMPER_5: Tamper5
  - RTC_TAMPER_6: Tamper6
  - RTC_TAMPER_7: Tamper7
  - RTC_TAMPER_8: Tamper8

__HAL_RTC_TAMPER_ENABLE_IT

Description:

- Enable the TAMP Tamper interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt

Return value:

- None

__HAL_RTC_TAMPER_DISABLE_IT

Description:
- Disable the TAMP Tamper interrupt.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt

Return value:
- None

__HAL_RTC_TAMPER_GET_IT

Description:
- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt
  - RTC_IT_INT_TAMP_ALL: All Internal Tamper interrupts
  - RTC_IT_INT_TAMP_1: Internal Tamper1 interrupt
  - RTC_IT_INT_TAMP_2: Internal Tamper2 interrupt
  - RTC_IT_INT_TAMP_3: Internal Tamper3 interrupt
  - RTC_IT_INT_TAMP_5: Internal Tamper5 interrupt
  - RTC_IT_INT_TAMP_6: Internal Tamper6 interrupt
  - RTC_IT_INT_TAMP_7: Internal Tamper7 interrupt
  - RTC_IT_INT_TAMP_8: Internal Tamper8 interrupt
  - RTC_IT_INT_TAMP_9: Internal Tamper9 interrupt
  - RTC_IT_INT_TAMP_11: Internal Tamper11 interrupt
  - RTC_IT_INT_TAMP_12: Internal Tamper12 interrupt
  - RTC_IT_INT_TAMP_13: Internal Tamper13 interrupt

Return value:
- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Tamper interrupt source to check. This parameter can be:
  - RTC_IT_TAMP_ALL: All tampers interrupts
  - RTC_IT_TAMP_1: Tamper1 interrupt
  - RTC_IT_TAMP_2: Tamper2 interrupt
  - RTC_IT_TAMP_3: Tamper3 interrupt
  - RTC_IT_TAMP_4: Tamper4 interrupt
  - RTC_IT_TAMP_5: Tamper5 interrupt
  - RTC_IT_TAMP_6: Tamper6 interrupt
  - RTC_IT_TAMP_7: Tamper7 interrupt
  - RTC_IT_TAMP_8: Tamper8 interrupt
  - RTC_IT_INT_TAMP_ALL: All internal tampers interrupts
  - RTC_IT_INT_TAMP_1: Internal Tamper1 interrupt
  - RTC_IT_INT_TAMP_2: Internal Tamper2 interrupt
  - RTC_IT_INT_TAMP_3: Internal Tamper3 interrupt
  - RTC_IT_INT_TAMP_5: Internal Tamper5 interrupt
  - RTC_IT_INT_TAMP_6: Internal Tamper6 interrupt
  - RTC_IT_INT_TAMP_7: Internal Tamper7 interrupt
  - RTC_IT_INT_TAMP_8: Internal Tamper8 interrupt
  - RTC_IT_INT_TAMP_9: Internal Tamper9 interrupt
  - RTC_IT_INT_TAMP_11: Internal Tamper11 interrupt
  - RTC_IT_INT_TAMP_12: Internal Tamper12 interrupt
  - RTC_IT_INT_TAMP_13: Internal Tamper13 interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE)

__HAL_RTC_TAMPER_GET_FLAG

Description:

- Get the selected RTC Tampers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag is pending or not. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag
  - RTC_FLAG_TAMP_3: Tamper3 flag
  - RTC_FLAG_TAMP_4: Tamper4 flag
  - RTC_FLAG_TAMP_5: Tamper5 flag
  - RTC_FLAG_TAMP_6: Tamper6 flag
  - RTC_FLAG_TAMP_7: Tamper7 flag
  - RTC_FLAG_TAMP_8: Tamper8 flag
  - RTC_FLAG_INT_TAMP_1: Internal Tamper1 flag
  - RTC_FLAG_INT_TAMP_2: Internal Tamper2 flag
  - RTC_FLAG_INT_TAMP_3: Internal Tamper3 flag
  - RTC_FLAG_INT_TAMP_5: Internal Tamper5 flag
  - RTC_FLAG_INT_TAMP_6: Internal Tamper6 flag
  - RTC_FLAG_INT_TAMP_7: Internal Tamper7 flag
  - RTC_FLAG_INT_TAMP_8: Internal Tamper8 flag
  - RTC_FLAG_INT_TAMP_9: Internal Tamper9 flag
  - RTC_FLAG_INT_TAMP_11: Internal Tamper11 flag
  - RTC_FLAG_INT_TAMP_12: Internal Tamper12 flag
  - RTC_FLAG_INT_TAMP_13: Internal Tamper13 flag

Return value:

- The: state of __FLAG__ (TRUE or FALSE)

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Tamper Flag to clear. This parameter can be:
  - RTC_FLAG_TAMP_ALL: All tampers flag
  - RTC_FLAG_TAMP_1: Tamper1 flag
  - RTC_FLAG_TAMP_2: Tamper2 flag
  - RTC_FLAG_TAMP_3: Tamper3 flag
  - RTC_FLAG_TAMP_4: Tamper4 flag
  - RTC_FLAG_TAMP_5: Tamper5 flag
  - RTC_FLAG_TAMP_6: Tamper6 flag
  - RTC_FLAG_TAMP_7: Tamper7 flag
  - RTC_FLAG_TAMP_8: Tamper8 flag
  - RTC_FLAG_INT_TAMP_ALL: All Internal Tamper flags
  - RTC_FLAG_INT_TAMP_1: Internal Tamper1 flag
  - RTC_FLAG_INT_TAMP_2: Internal Tamper2 flag
  - RTC_FLAG_INT_TAMP_3: Internal Tamper3 flag
  - RTC_FLAG_INT_TAMP_5: Internal Tamper5 flag
  - RTC_FLAG_INT_TAMP_6: Internal Tamper6 flag
  - RTC_FLAG_INT_TAMP_7: Internal Tamper7 flag
  - RTC_FLAG_INT_TAMP_8: Internal Tamper8 flag
  - RTC_FLAG_INT_TAMP_9: Internal Tamper9 flag
  - RTC_FLAG_INT_TAMP_11: Internal Tamper11 flag
  - RTC_IT_INT_TAMP_12: Internal Tamper12 interrupt
  - RTC_IT_INT_TAMP_13: Internal Tamper13 interrupt

Return value:

- None

RTCEx Tamper EraseBackUp

RTC_TAMPER_ERASE_BACKUP_ENABLE

RTC_TAMPER_ERASE_BACKUP_DISABLE

RTCEx Tamper Filter

RTC_TAMPERFILTER_DISABLE

Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level

RTCEx Tamper MaskFlag

RTC_TAMPERMASK_FLAG_DISABLE

RTC_TAMPERMASK_FLAG_ENABLE

RTCEx Tamper Pins Definition

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

RTC_TAMPER_4

RTC_TAMPER_5

RTC_TAMPER_6

RTC_TAMPER_7

RTC_TAMPER_8

RTC_TAMPER_ALL

RTCEx Tamper Pin Precharge Duration

RTC_TAMPERPRECHARGEDURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

RTCEx Tamper Pull UP

RTC_TAMPER_PULLUP_ENABLE

Tamper pins are pre-charged before sampling

RTC_TAMPER_PULLUP_DISABLE

Tamper pins pre-charge is disabled

RTCEx Tamper Sampling Frequencies

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768

Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384

Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192

Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096

Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048

Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024

Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512

Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256

Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

RTCEx Tamper TimeStamp On Tamper Detection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE

TimeStamp on Tamper Detection event is not saved

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE

TimeStamp on Tamper Detection event saved

RTCEx Tamper Trigger

RTC_TAMPERTRIGGER_RISINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_FALLINGEDGE

Warning : Filter must be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_LOWLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTC_TAMPERTRIGGER_HIGHLEVEL

Warning : Filter must not be RTC_TAMPERFILTER_DISABLE

RTCEx TAMP Monotonic Counter Privilege Definition

TAMP_MONOTONIC_CNT_PRIVILEGE_YES

TAMP_MONOTONIC_CNT_PRIVILEGE_NO

RTCEx TAMP Monotonic Counter Secure Definition

TAMP_MONOTONIC_CNT_SECURE_YES

TAMPER Monotonic Counter secure

TAMP_MONOTONIC_CNT_SECURE_NO

TAMPER Monotonic Counter is not secure

RTCEx TAMP security Definition

TAMP_PRIVILEGE_FULL_YES

TAMP_PRIVILEGE_FULL_NO

RTCEx TAMP Secret Device Conf Erase Definition

TAMP_SECRETDEVICE_ERASE_ENABLE

TAMP_SECRETDEVICE_ERASE_DISABLE

RTCEx TAMP Secure Definition

TAMP_SECURE_FULL_YES

TAMPER full secure

TAMP_SECURE_FULL_NO

TAMPER is not secure

RTC Timestamp

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE_IT

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be enabled. This parameter can be:
  – RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE_IT

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt source to be disabled. This parameter can be:
  – RTC_IT_TS TimeStamp interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:
- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt to check. This parameter can be:
  – RTC_IT_TS TimeStamp interrupt

Return value:
- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_TIMESTAMP_GET_IT_SOURCE

Description:
- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
  – RTC_IT_TS TimeStamp interrupt

Return value:
- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_TIMESTAMP_GET_FLAG

Description:
- Get the selected RTC TimeStamps flag status.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag is pending or not. This parameter can be:
  – RTC_FLAG_TSF
  – RTC_FLAG_TSOVF

Return value:
- The: state of __FLAG__ (TRUE or FALSE) or 255 if invalid parameter.

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:
- Clear the RTC Time Stamps pending flags.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC TimeStamp Flag to clear. This parameter can be:
  – RTC_FLAG_TSF
  – RTC_FLAG_TSOVF

Return value:
- None

__HAL_RTC_INTERNAL_TIMESTAMP_ENABLE

Description:
- Enable the RTC internal TimeStamp peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_INTERNAL_TIMESTAMP_DISABLE

Description:
- Disable the RTC internal TimeStamp peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_INTERNAL_TIMESTAMP_GET_FLAG

Description:
- Get the selected RTC Internal Time Stamps flag status.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag is pending or not. This parameter can be:
  – RTC_FLAG_ITSF

Return value:
- None

__HAL_RTC_INTERNAL_TIMESTAMP_CLEAR_FLAG

Description:
- Clear the RTC Internal Time Stamps pending flags.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Internal Time Stamp Flag source to clear. This parameter can be:
  – RTC_FLAG_ITSF

Return value:
- None

__HAL_RTC_TAMPTS_ENABLE

Description:
- Enable the RTC TimeStamp on Tamper detection.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_TAMPTS_DISABLE

Description:
- Disable the RTC TimeStamp on Tamper detection.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_TAMPOE_ENABLE

Description:
- Enable the RTC Tamper detection output.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_TAMPOE_DISABLE

Description:
- Disable the RTC Tamper detection output.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

RTC WakeUp Timer

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:
- Enable the RTC WakeUp Timer peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:
- Disable the RTC WakeUp Timer peripheral.

Parameters:
- __HANDLE__: specifies the RTC handle.

Return value:
- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be enabled. This parameter can be:
  – RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt sources to be disabled. This parameter can be:
  – RTC_IT_WUT WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC WakeUpTimer interrupt to check. This parameter can be:
  – RTC_IT_WUT WakeUpTimer interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
  – RTC_IT_WUT WakeUpTimer interrupt

Return value:

- The: state of __INTERRUPT__ (TRUE or FALSE).

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:

- Get the selected RTC WakeUpTimers flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag is pending or not. This parameter can be:
  - RTC_FLAG_WUTF
  - RTC_FLAG_WUTWF

Return value:

- The: state of __FLAG__ (TRUE or FALSE).

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:

- Clear the RTC Wake Up timers pending flags.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC WakeUpTimer Flag to clear. This parameter can be:
  - RTC_FLAG_WUTF

Return value:

- None

RTCEx Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

70 HAL SAI Generic Driver

70.1 SAI Firmware driver registers structures

70.1.1 SAI_PdmInitTypeDef

SAI_PdmInitTypeDef is defined in the stm32u5xx_hal_sai.h

Data Fields

- ***FunctionalState Activation***
- ***uint32_t MicPairsNbr***
- ***uint32_t ClockEnable***

Field Documentation

- ***FunctionalState SAI_PdmInitTypeDef::Activation***
  Enable/disable PDM interface
- ***uint32_t SAI_PdmInitTypeDef::MicPairsNbr***
  Specifies the number of microphone pairs used. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAI_PdmInitTypeDef::ClockEnable***
  Specifies which clock must be enabled. This parameter can be a values combination of ***SAI_PDM_ClockEnable***

70.1.2 SAI_InitTypeDef

SAI_InitTypeDef is defined in the stm32u5xx_hal_sai.h

Data Fields

- ***uint32_t AudioMode***
- ***uint32_t Synchro***
- ***uint32_t SynchroExt***
- ***uint32_t MckOutput***
- ***uint32_t OutputDrive***
- ***uint32_t NoDivider***
- ***uint32_t FIFOThreshold***
- ***uint32_t AudioFrequency***
- ***uint32_t Mckdiv***
- ***uint32_t MckOverSampling***
- ***uint32_t MonoStereoMode***
- ***uint32_t CompandingMode***
- ***uint32_t TriState***
- ***SAI_PdmInitTypeDef PdmInit***
- ***uint32_t Protocol***
- ***uint32_t DataSize***
- ***uint32_t FirstBit***
- ***uint32_t ClockStrobing***

Field Documentation

- ***uint32_t SAI_InitTypeDef::AudioMode***
  Specifies the SAI Block audio Mode. This parameter can be a value of ***SAI_Block_Mode***
- ***uint32_t SAI_InitTypeDef::Synchro***
  Specifies SAI Block synchronization This parameter can be a value of ***SAI_Block_Synchronization***

- ***uint32_t SAI_InitTypeDef::SynchroExt***
  Specifies SAI external output synchronization, this setup is common for BlockA and BlockB This parameter can be a value of ***SAI_Block_SyncExt***
  Note:
  – If both audio blocks of same SAI are used, this parameter has to be set to the same value for each audio block
- ***uint32_t SAI_InitTypeDef::MckOutput***
  Specifies whether master clock output will be generated or not. This parameter can be a value of ***SAI_Block_MckOutput***
- ***uint32_t SAI_InitTypeDef::OutputDrive***
  Specifies when SAI Block outputs are driven. This parameter can be a value of ***SAI_Block_Output_Drive***
  Note:
  – This value has to be set before enabling the audio block but after the audio block configuration.
- ***uint32_t SAI_InitTypeDef::NoDivider***
  Specifies whether master clock will be divided or not. This parameter can be a value of ***SAI_Block_NoDivider***
  Note:
  – If bit NODIV in the SAI_xCR1 register is cleared, the frame length should be aligned to a number equal to a power of 2, from 8 to 256. If bit NODIV in the SAI_xCR1 register is set, the frame length can take any of the values from 8 to 256.
- ***uint32_t SAI_InitTypeDef::FIFOThreshold***
  Specifies SAI Block FIFO threshold. This parameter can be a value of ***SAI_Block_Fifo_Threshold***
- ***uint32_t SAI_InitTypeDef::AudioFrequency***
  Specifies the audio frequency sampling. This parameter can be a value of ***SAI_Audio_Frequency***
- ***uint32_t SAI_InitTypeDef::Mckdiv***
  Specifies the master clock divider. This parameter must be a number between Min_Data = 0 and Max_Data = 63.
  Note:
  – This parameter is used only if AudioFrequency is set to SAI_AUDIO_FREQUENCY_MCKDIV otherwise it is internally computed.
- ***uint32_t SAI_InitTypeDef::MckOverSampling***
  Specifies the master clock oversampling. This parameter can be a value of ***SAI_Block_Mck_OverSampling***
- ***uint32_t SAI_InitTypeDef::MonoStereoMode***
  Specifies if the mono or stereo mode is selected. This parameter can be a value of ***SAI_Mono_Stereo_Mode***
- ***uint32_t SAI_InitTypeDef::CompandingMode***
  Specifies the companding mode type. This parameter can be a value of ***SAI_Block_Companding_Mode***
- ***uint32_t SAI_InitTypeDef::TriState***
  Specifies the companding mode type. This parameter can be a value of ***SAI_TRIState_Management***
- ***SAI_PdmInitTypeDef SAI_InitTypeDef::PdmInit***
  Specifies the PDM configuration.
- ***uint32_t SAI_InitTypeDef::Protocol***
  Specifies the SAI Block protocol. This parameter can be a value of ***SAI_Block_Protocol***
- ***uint32_t SAI_InitTypeDef::DataSize***
  Specifies the SAI Block data size. This parameter can be a value of ***SAI_Block_Data_Size***
- ***uint32_t SAI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SAI_Block_MSB_LSB_transmission***
- ***uint32_t SAI_InitTypeDef::ClockStrobing***
  Specifies the SAI Block clock strobing edge sensitivity. This parameter can be a value of ***SAI_Block_Clock_Strobing***

70.1.3 SAI_FrameInitTypeDef

SAI_FrameInitTypeDef is defined in the stm32u5xx_hal_sai.h

Data Fields

- *uint32_t FrameLength*
- *uint32_t ActiveFrameLength*
- *uint32_t FSDefinition*
- *uint32_t FSPolarity*
- *uint32_t FSOffset*

Field Documentation

- ***uint32_t SAI_FrameInitTypeDef::FrameLength***
  Specifies the Frame length, the number of SCK clocks for each audio frame. This parameter must be a number between Min_Data = 8 and Max_Data = 256.
  Note:
  – If master clock MCLK_x pin is declared as an output, the frame length should be aligned to a number equal to power of 2 in order to keep in an audio frame, an integer number of MCLK pulses by bit Clock.
- ***uint32_t SAI_FrameInitTypeDef::ActiveFrameLength***
  Specifies the Frame synchronization active level length. This Parameter specifies the length in number of bit clock (SCK + 1) of the active level of FS signal in audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 128
- ***uint32_t SAI_FrameInitTypeDef::FSDefinition***
  Specifies the Frame synchronization definition. This parameter can be a value of ***SAI_Block_FS_Definition***
- ***uint32_t SAI_FrameInitTypeDef::FSPolarity***
  Specifies the Frame synchronization Polarity. This parameter can be a value of ***SAI_Block_FS_Polarity***
- ***uint32_t SAI_FrameInitTypeDef::FSOffset***
  Specifies the Frame synchronization Offset. This parameter can be a value of ***SAI_Block_FS_Offset***

70.1.4 SAI_SlotInitTypeDef

SAI_SlotInitTypeDef is defined in the stm32u5xx_hal_sai.h

Data Fields

- *uint32_t FirstBitOffset*
- *uint32_t SlotSize*
- *uint32_t SlotNumber*
- *uint32_t SlotActive*

Field Documentation

- ***uint32_t SAI_SlotInitTypeDef::FirstBitOffset***
  Specifies the position of first data transfer bit in the slot. This parameter must be a number between Min_Data = 0 and Max_Data = 24
- ***uint32_t SAI_SlotInitTypeDef::SlotSize***
  Specifies the Slot Size. This parameter can be a value of ***SAI_Block_Slot_Size***
- ***uint32_t SAI_SlotInitTypeDef::SlotNumber***
  Specifies the number of slot in the audio frame. This parameter must be a number between Min_Data = 1 and Max_Data = 16
- ***uint32_t SAI_SlotInitTypeDef::SlotActive***
  Specifies the slots in audio frame that will be activated. This parameter can be a value of ***SAI_Block_Slot_Active***

70.1.5 __SAI_HandleTypeDef

__SAI_HandleTypeDef is defined in the stm32u5xx_hal_sai.h

Data Fields

- *SAI_Block_TypeDef * Instance*
- *SAI_InitTypeDef Init*
- *SAI_FrameInitTypeDef FrameInit*
- *SAI_SlotInitTypeDef SlotInit*
- *uint8_t * pBuffPtr*
- *uint16_t XferSize*

- ***uint16_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***SAIcallback mutecallback***
- ***void(* InterruptServiceRoutine***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SAI_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* RxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***SAI_Block_TypeDef* __SAI_HandleTypeDef::Instance***
  SAI Blockx registers base address
- ***SAI_InitTypeDef __SAI_HandleTypeDef::Init***
  SAI communication parameters
- ***SAI_FrameInitTypeDef __SAI_HandleTypeDef::FrameInit***
  SAI Frame configuration parameters
- ***SAI_SlotInitTypeDef __SAI_HandleTypeDef::SlotInit***
  SAI Slot configuration parameters
- ***uint8_t* __SAI_HandleTypeDef::pBuffPtr***
  Pointer to SAI transfer Buffer
- ***uint16_t __SAI_HandleTypeDef::XferSize***
  SAI transfer size
- ***uint16_t __SAI_HandleTypeDef::XferCount***
  SAI transfer counter
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmatx***
  SAI Tx DMA handle parameters
- ***DMA_HandleTypeDef* __SAI_HandleTypeDef::hdmarx***
  SAI Rx DMA handle parameters
- ***SAIcallback __SAI_HandleTypeDef::mutecallback***
  SAI mute callback
- ***void(* __SAI_HandleTypeDef::InterruptServiceRoutine)(struct __SAI_HandleTypeDef *hsai)***
- ***HAL_LockTypeDef __SAI_HandleTypeDef::Lock***
  SAI locking object
- ***__IO HAL_SAI_StateTypeDef __SAI_HandleTypeDef::State***
  SAI communication state
- ***__IO uint32_t __SAI_HandleTypeDef::ErrorCode***
  SAI Error code
- ***void(* __SAI_HandleTypeDef::RxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive complete callback
- ***void(* __SAI_HandleTypeDef::RxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI receive half complete callback
- ***void(* __SAI_HandleTypeDef::TxCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit complete callback
- ***void(* __SAI_HandleTypeDef::TxHalfCpltCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI transmit half complete callback
- ***void(* __SAI_HandleTypeDef::ErrorCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI error callback

- ***void(* __SAI_HandleTypeDef::MspInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP init callback
- ***void(* __SAI_HandleTypeDef::MspDeInitCallback)(struct __SAI_HandleTypeDef *hsai)***
  SAI MSP de-init callback

70.2 SAI Firmware driver API description

The following section lists the various functions of the SAI library.

70.2.1 How to use this driver

The SAI HAL driver can be used as follows:

1. Declare a SAI_HandleTypeDef handle structure (eg. SAI_HandleTypeDef hsai).
2. Initialize the SAI low level resources by implementing the HAL_SAI_MspInit() API:
   a. Enable the SAI interface clock.
   b. SAI pins configuration:
      - Enable the clock for the SAI GPIOs.
      - Configure these SAI pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_SAI_Transmit_IT() and HAL_SAI_Receive_IT() APIs):
      - Configure the SAI interrupt priority.
      - Enable the NVIC SAI IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_SAI_Transmit_DMA() and HAL_SAI_Receive_DMA() APIs):
      - Declare a DMA handle structure for the Tx/Rx stream.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream.
      - Associate the initialized DMA handle to the SAI DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream.
3. The initialization can be done by two ways
   a. Expert mode : Initialize the structures Init, FrameInit and SlotInit and call HAL_SAI_Init().
   b. Simplified mode : Initialize the high part of Init Structure and call HAL_SAI_InitProtocol().

Note: *The specific SAI interrupts (FIFO request and Overrun underrun interrupt) will be managed using the macros __HAL_SAI_ENABLE_IT() and __HAL_SAI_DISABLE_IT() inside the transmit and receive process.*

Note: *Make sure that either:*

- *PLLSAI1CLK output is configured or*
- *PLLSAI2CLK output is configured or*
- *PLLSAI3CLK output is configured or*
- *External clock source is configured after setting correctly the define constant EXTERNAL_SAI1_CLOCK_VALUE or EXTERNAL_SAI2_CLOCK_VALUE in the stm32u5xx_hal_conf.h file.*

Note: *In master Tx mode: enabling the audio block immediately generates the bit clock for the external slaves even if there is no data in the FIFO, However FS signal generation is conditioned by the presence of data in the FIFO.*

Note: *In master Rx mode: enabling the audio block immediately generates the bit clock and FS signal for the external slaves.*

Note: *It is mandatory to respect the following conditions in order to avoid bad SAI behavior:*

- *First bit Offset <= (SLOT size - Data size)*
- *Data size <= SLOT size*
- *Number of SLOT x SLOT size = Frame length*
- *The number of slots should be even when SAI_FS_CHANNEL_IDENTIFICATION is selected.*

Note: *PDM interface can be activated through HAL_SAI_Init function. Please note that PDM interface is only available for SAI1 sub-block A. PDM microphone delays can be tuned with HAL_SAIEx_ConfigPdmMicDelay function.*

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SAI_Transmit()
- Receive an amount of data in blocking mode using HAL_SAI_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SAI_Transmit_IT()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SAI_Receive_IT()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SAI_Transmit_DMA()
- At transmission end of transfer HAL_SAI_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SAI_Receive_DMA()
- At reception end of transfer HAL_SAI_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SAI_RxCpltCallback()
- In case of flag error, HAL_SAI_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SAI_ErrorCallback()
- Pause the DMA Transfer using HAL_SAI_DMAPause()
- Resume the DMA Transfer using HAL_SAI_DMAResume()
- Stop the DMA Transfer using HAL_SAI_DMAStop()

SAI HAL driver additional function list

Below the list the others API available SAI HAL driver :

- HAL_SAI_EnableTxMuteMode(): Enable the mute in tx mode
- HAL_SAI_DisableTxMuteMode(): Disable the mute in tx mode
- HAL_SAI_EnableRxMuteMode(): Enable the mute in Rx mode
- HAL_SAI_DisableRxMuteMode(): Disable the mute in Rx mode
- HAL_SAI_FlushRxFifo(): Flush the rx fifo.
- HAL_SAI_Abort(): Abort the current transfer

SAI HAL driver macros list

Below the list of most used macros in SAI HAL driver :

- __HAL_SAI_ENABLE(): Enable the SAI peripheral
- __HAL_SAI_DISABLE(): Disable the SAI peripheral
- __HAL_SAI_ENABLE_IT(): Enable the specified SAI interrupts
- __HAL_SAI_DISABLE_IT(): Disable the specified SAI interrupts
- __HAL_SAI_GET_IT_SOURCE(): Check if the specified SAI interrupt source is enabled or disabled
- __HAL_SAI_GET_FLAG(): Check whether the specified SAI flag is set or not

Callback registration

The compilation define USE_HAL_SAI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use functions HAL_SAI_RegisterCallback() to register a user callback.

Function HAL_SAI_RegisterCallback() allows to register following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

This function takes as parameters the HAL peripheral handle, the callback ID and a pointer to the user callback function.

Use function HAL_SAI_UnRegisterCallback() to reset a callback to the default weak function. HAL_SAI_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the callback ID.

This function allows to reset following callbacks:

- RxCpltCallback : SAI receive complete.
- RxHalfCpltCallback : SAI receive half complete.
- TxCpltCallback : SAI transmit complete.
- TxHalfCpltCallback : SAI transmit half complete.
- ErrorCallback : SAI error.
- MspInitCallback : SAI MspInit.
- MspDeInitCallback : SAI MspDeInit.

By default, after the HAL_SAI_Init and if the state is HAL_SAI_STATE_RESET all callbacks are reset to the corresponding legacy weak functions: examples HAL_SAI_RxCpltCallback(), HAL_SAI_ErrorCallback(). Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak functions in the HAL_SAI_Init and HAL_SAI_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SAI_Init and HAL_SAI_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SAI_RegisterCallback before calling HAL_SAI_DeInit or HAL_SAI_Init function.

When the compilation define USE_HAL_SAI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used.

70.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SAIx peripheral:

- User must implement HAL_SAI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SAI_Init() to configure the selected device with the selected configuration:
  – Mode (Master/slave TX/RX)
  – Protocol
  – Data Size
  – MCLK Output
  – Audio frequency
  – FIFO Threshold
  – Frame Config
  – Slot Config
  – PDM Config
- Call the function HAL_SAI_DeInit() to restore the default configuration of the selected SAI peripheral.

This section contains the following APIs:

- *HAL_SAI_InitProtocol()*
- *HAL_SAI_Init()*
- *HAL_SAI_DeInit()*

- *HAL_SAI_MspInit()*
- *HAL_SAI_MspDeInit()*
- *HAL_SAI_RegisterCallback()*
- *HAL_SAI_UnRegisterCallback()*

70.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SAI data transfers.

- There are two modes of transfer:
  - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
  - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SAI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
- Blocking mode functions are :
  - HAL_SAI_Transmit()
  - HAL_SAI_Receive()
- Non Blocking mode functions with Interrupt are :
  - HAL_SAI_Transmit_IT()
  - HAL_SAI_Receive_IT()
- Non Blocking mode functions with DMA are :
  - HAL_SAI_Transmit_DMA()
  - HAL_SAI_Receive_DMA()
- A set of Transfer Complete Callbacks are provided in non Blocking mode:
  - HAL_SAI_TxCpltCallback()
  - HAL_SAI_RxCpltCallback()
  - HAL_SAI_ErrorCallback()

This section contains the following APIs:

- *HAL_SAI_Transmit()*
- *HAL_SAI_Receive()*
- *HAL_SAI_Transmit_IT()*
- *HAL_SAI_Receive_IT()*
- *HAL_SAI_DMAPause()*
- *HAL_SAI_DMAResume()*
- *HAL_SAI_DMAStop()*
- *HAL_SAI_Abort()*
- *HAL_SAI_Transmit_DMA()*
- *HAL_SAI_Receive_DMA()*
- *HAL_SAI_EnableTxMuteMode()*
- *HAL_SAI_DisableTxMuteMode()*
- *HAL_SAI_EnableRxMuteMode()*
- *HAL_SAI_DisableRxMuteMode()*
- *HAL_SAI_IRQHandler()*
- *HAL_SAI_TxCpltCallback()*
- *HAL_SAI_TxHalfCpltCallback()*
- *HAL_SAI_RxCpltCallback()*
- *HAL_SAI_RxHalfCpltCallback()*
- *HAL_SAI_ErrorCallback()*

70.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:
- *HAL_SAI_GetState()*
- *HAL_SAI_GetError()*

70.2.5 Detailed description of functions

HAL_SAI_InitProtocol

Function name

HAL_StatusTypeDef HAL_SAI_InitProtocol (SAI_HandleTypeDef * hsai, uint32_t protocol, uint32_t datasize, uint32_t nbslot)

Function description

Initialize the structure FrameInit, SlotInit and the low part of Init according to the specified parameters and call the function HAL_SAI_Init to initialize the SAI block.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **protocol:** one of the supported protocol SAI Supported protocol
- **datasize:** one of the supported datasize SAI protocol data size the configuration information for SAI module.
- **nbslot:** Number of slot.

Return values

- **HAL:** status

HAL_SAI_Init

Function name

HAL_StatusTypeDef HAL_SAI_Init (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI according to the specified parameters.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DeInit

Function name

HAL_StatusTypeDef HAL_SAI_DeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI peripheral.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_MspInit

Function name

void HAL_SAI_MspInit (SAI_HandleTypeDef * hsai)

Function description

Initialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_MspDeInit

Function name

void HAL_SAI_MspDeInit (SAI_HandleTypeDef * hsai)

Function description

DeInitialize the SAI MSP.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_RegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID, pSAI_CallbackTypeDef pCallback)

Function description

Register a user SAI callback to be used instead of the weak predefined callback.

Parameters

- **hsai:** SAI handle.
- **CallbackID:** ID of the callback to be registered. This parameter can be one of the following values:
  – HAL_SAI_RX_COMPLETE_CB_ID receive complete callback ID.
  – HAL_SAI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  – HAL_SAI_TX_COMPLETE_CB_ID transmit complete callback ID.
  – HAL_SAI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  – HAL_SAI_ERROR_CB_ID error callback ID.
  – HAL_SAI_MSPINIT_CB_ID MSP init callback ID.
  – HAL_SAI_MSPDEINIT_CB_ID MSP de-init callback ID.
- **pCallback:** pointer to the callback function.

Return values

- **HAL:** status.

HAL_SAI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SAI_UnRegisterCallback (SAI_HandleTypeDef * hsai, HAL_SAI_CallbackIDTypeDef CallbackID)

Function description

Unregister a user SAI callback.

Parameters

- **hsai:** SAI handle.
- **CallbackID:** ID of the callback to be unregistered. This parameter can be one of the following values:
  – HAL_SAI_RX_COMPLETE_CB_ID receive complete callback ID.
  – HAL_SAI_RX_HALFCOMPLETE_CB_ID receive half complete callback ID.
  – HAL_SAI_TX_COMPLETE_CB_ID transmit complete callback ID.
  – HAL_SAI_TX_HALFCOMPLETE_CB_ID transmit half complete callback ID.
  – HAL_SAI_ERROR_CB_ID error callback ID.
  – HAL_SAI_MSPINIT_CB_ID MSP init callback ID.
  – HAL_SAI_MSPDEINIT_CB_ID MSP de-init callback ID.

Return values

- **HAL:** status.

HAL_SAI_Transmit

Function name

HAL_StatusTypeDef HAL_SAI_Transmit (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Receive

Function name

HAL_StatusTypeDef HAL_SAI_Receive (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SAI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SAI_Receive_IT (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Transmit_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- **HAL:** status

HAL_SAI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SAI_Receive_DMA (SAI_HandleTypeDef * hsai, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be received

Return values

- **HAL:** status

HAL_SAI_DMAPause

Function name

HAL_StatusTypeDef HAL_SAI_DMAPause (SAI_HandleTypeDef * hsai)

Function description

Pause the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAResume

Function name

HAL_StatusTypeDef HAL_SAI_DMAResume (SAI_HandleTypeDef * hsai)

Function description

Resume the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_DMAStop

Function name

HAL_StatusTypeDef HAL_SAI_DMAStop (SAI_HandleTypeDef * hsai)

Function description

Stop the audio stream playing from the Media.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_Abort

Function name

HAL_StatusTypeDef HAL_SAI_Abort (SAI_HandleTypeDef * hsai)

Function description

Abort the current transfer and disable the SAI.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableTxMuteMode (SAI_HandleTypeDef * hsai, uint16_t val)

Function description

Enable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **val:** value sent during the mute SAI Block Mute Value

Return values

- **HAL:** status

HAL_SAI_DisableTxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableTxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Tx mute mode.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_EnableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_EnableRxMuteMode (SAI_HandleTypeDef * hsai, SAIcallback callback, uint16_t counter)

Function description

Enable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.
- **callback:** function called when the mute is detected.
- **counter:** number a data before mute detection max 63.

Return values

- **HAL:** status

HAL_SAI_DisableRxMuteMode

Function name

HAL_StatusTypeDef HAL_SAI_DisableRxMuteMode (SAI_HandleTypeDef * hsai)

Function description

Disable the Rx mute detection.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** status

HAL_SAI_IRQHandler

Function name

void HAL_SAI_IRQHandler (SAI_HandleTypeDef * hsai)

Function description

Handle SAI interrupt request.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxHalfCpltCallback

Function name

void HAL_SAI_TxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer Half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_TxCpltCallback

Function name

void HAL_SAI_TxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Tx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxHalfCpltCallback

Function name

void HAL_SAI_RxHalfCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer half completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_RxCpltCallback

Function name

void HAL_SAI_RxCpltCallback (SAI_HandleTypeDef * hsai)

Function description

Rx Transfer completed callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_ErrorCallback

Function name

void HAL_SAI_ErrorCallback (SAI_HandleTypeDef * hsai)

Function description

SAI error callback.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **None:**

HAL_SAI_GetState

Function name

HAL_SAI_StateTypeDef HAL_SAI_GetState (const SAI_HandleTypeDef * hsai)

Function description

Return the SAI handle state.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for SAI module.

Return values

- **HAL:** state

HAL_SAI_GetError

Function name

uint32_t HAL_SAI_GetError (const SAI_HandleTypeDef * hsai)

Function description

Return the SAI error code.

Parameters

- **hsai:** pointer to a SAI_HandleTypeDef structure that contains the configuration information for the specified SAI Block.

Return values

- **SAI:** Error Code

70.3 SAI Firmware driver defines

The following section lists the various define and macros of the module.

70.3.1 SAI

SAI

SAI Audio Frequency

SAI_AUDIO_FREQUENCY_192K

SAI_AUDIO_FREQUENCY_96K

SAI_AUDIO_FREQUENCY_48K

SAI_AUDIO_FREQUENCY_44K

SAI_AUDIO_FREQUENCY_32K

SAI_AUDIO_FREQUENCY_22K

SAI_AUDIO_FREQUENCY_16K

SAI_AUDIO_FREQUENCY_11K

SAI_AUDIO_FREQUENCY_8K

SAI_AUDIO_FREQUENCY_MCKDIV

SAI Block Clock Strobing

SAI_CLOCKSTROBING_FALLINGEDGE

SAI_CLOCKSTROBING_RISINGEDGE

SAI Block Companding Mode

SAI_NOCOMPANDING

SAI_ULAW_1CPL_COMPANDING

SAI_ALAW_1CPL_COMPANDING

SAI_ULAW_2CPL_COMPANDING

SAI_ALAW_2CPL_COMPANDING

SAI Block Data Size

SAI_DATASIZE_8

SAI_DATASIZE_10

SAI_DATASIZE_16

SAI_DATASIZE_20

SAI_DATASIZE_24

SAI_DATASIZE_32

SAI Block Fifo Status Level

SAI_FIFOSTATUS_EMPTY

SAI_FIFOSTATUS_LESS1QUARTERFULL

SAI_FIFOSTATUS_1QUARTERFULL

SAI_FIFOSTATUS_HALFFULL

SAI_FIFOSTATUS_3QUARTERFULL

SAI_FIFOSTATUS_FULL

SAI Block Fifo Threshold

SAI_FIFOTHRESHOLD_EMPTY

SAI_FIFOTHRESHOLD_1QF

SAI_FIFOTHRESHOLD_HF

SAI_FIFOTHRESHOLD_3QF

SAI_FIFOTHRESHOLD_FULL

SAI Block Flags Definition

SAI_FLAG_OVRUDR

SAI_FLAG_MUTEDET

SAI_FLAG_WCKCFG

SAI_FLAG_FREQ

SAI_FLAG_CNRDY

SAI_FLAG_AFSDET

SAI_FLAG_LFSDET

SAI Block FS Definition

SAI_FS_STARTFRAME

SAI_FS_CHANNEL_IDENTIFICATION

SAI Block FS Offset

SAI_FS_FIRSTBIT

SAI_FS_BEFOREFIRSTBIT

SAI Block FS Polarity

SAI_FS_ACTIVE_LOW

SAI_FS_ACTIVE_HIGH

SAI Block Interrupts Definition

SAI_IT_OVRUDR

SAI_IT_MUTEDET

SAI_IT_WCKCFG

SAI_IT_FREQ

SAI_IT_CNRDY

SAI_IT_AFSDET

SAI_IT_LFSDET

SAI Block Master Clock Output

SAI_MCK_OUTPUT_DISABLE

SAI_MCK_OUTPUT_ENABLE

SAI Block Master Clock OverSampling

SAI_MCK_OVERSAMPLING_DISABLE

SAI_MCK_OVERSAMPLING_ENABLE

SAI Block Mode

SAI_MODEMASTER_TX

SAI_MODEMASTER_RX

SAI_MODESLAVE_TX

SAI_MODESLAVE_RX

SAI Block MSB LSB transmission

SAI_FIRSTBIT_MSB

SAI_FIRSTBIT_LSB

SAI Block Mute Value

SAI_ZERO_VALUE

SAI_LAST_SENT_VALUE

SAI Block NoDivider

SAI_MASTERDIVIDER_ENABLE

SAI_MASTERDIVIDER_DISABLE

SAI Block Output Drive

SAI_OUTPUTDRIVE_DISABLE

SAI_OUTPUTDRIVE_ENABLE

SAI Block Protocol

SAI_FREE_PROTOCOL

SAI_SPDIF_PROTOCOL

SAI_AC97_PROTOCOL

SAI Block Slot Active

SAI_SLOT_NOTACTIVE

SAI_SLOTACTIVE_0

SAI_SLOTACTIVE_1

SAI_SLOTACTIVE_2

SAI_SLOTACTIVE_3

SAI_SLOTACTIVE_4

SAI_SLOTACTIVE_5

SAI_SLOTACTIVE_6

SAI_SLOTACTIVE_7

SAI_SLOTACTIVE_8

SAI_SLOTACTIVE_9

SAI_SLOTACTIVE_10

SAI_SLOTACTIVE_11

SAI_SLOTACTIVE_12

SAI_SLOTACTIVE_13

SAI_SLOTACTIVE_14

SAI_SLOTACTIVE_15

SAI_SLOTACTIVE_ALL

SAI Block Slot Size

SAI_SLOTSIZE_DATASIZE

SAI_SLOTSIZE_16B

SAI_SLOTSIZE_32B

SAI External synchronisation

SAI_SYNCEXT_DISABLE

SAI_SYNCEXT_OUTBLOCKA_ENABLE

SAI_SYNCEXT_OUTBLOCKB_ENABLE

SAI Block Synchronization

SAI_ASYNCHRONOUS

Asynchronous

SAI_SYNCHRONOUS

Synchronous with other block of same SAI

SAI_SYNCHRONOUS_EXT_SAI1

Synchronous with other SAI, SAI1

SAI_SYNCHRONOUS_EXT_SAI2

Synchronous with other SAI, SAI2

SAI Error Code

HAL_SAI_ERROR_NONE

No error

HAL_SAI_ERROR_OVR

Overrun Error

HAL_SAI_ERROR_UDR

Underrun error

HAL_SAI_ERROR_AFSDET

Anticipated Frame synchronisation detection

HAL_SAI_ERROR_LFSDET

Late Frame synchronisation detection

HAL_SAI_ERROR_CNREADY

codec not ready

HAL_SAI_ERROR_WCKCFG

Wrong clock configuration

HAL_SAI_ERROR_TIMEOUT

Timeout error

HAL_SAI_ERROR_DMA

DMA error

HAL_SAI_ERROR_INVALID_CALLBACK

Invalid callback error

SAI Exported Macros

__HAL_SAI_RESET_HANDLE_STATE

Description:

- Reset SAI handle state.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_ENABLE_IT

Description:

- Enable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  – SAI_IT_OVRUDR: Overrun underrun interrupt enable
  – SAI_IT_MUTEDET: Mute detection interrupt enable
  – SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  – SAI_IT_FREQ: FIFO request interrupt enable
  – SAI_IT_CNRDY: Codec not ready interrupt enable
  – SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  – SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_DISABLE_IT

Description:

- Disable the specified SAI interrupts.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  – SAI_IT_OVRUDR: Overrun underrun interrupt enable
  – SAI_IT_MUTEDET: Mute detection interrupt enable
  – SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  – SAI_IT_FREQ: FIFO request interrupt enable
  – SAI_IT_CNRDY: Codec not ready interrupt enable
  – SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  – SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- None

__HAL_SAI_GET_IT_SOURCE

Description:

- Check whether the specified SAI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __INTERRUPT__: specifies the SAI interrupt source to check. This parameter can be one of the following values:
  – SAI_IT_OVRUDR: Overrun underrun interrupt enable
  – SAI_IT_MUTEDET: Mute detection interrupt enable
  – SAI_IT_WCKCFG: Wrong Clock Configuration interrupt enable
  – SAI_IT_FREQ: FIFO request interrupt enable
  – SAI_IT_CNRDY: Codec not ready interrupt enable
  – SAI_IT_AFSDET: Anticipated frame synchronization detection interrupt enable
  – SAI_IT_LFSDET: Late frame synchronization detection interrupt enable

Return value:

- The: new state of __INTERRUPT__ (TRUE or FALSE).

__HAL_SAI_GET_FLAG

Description:

- Check whether the specified SAI flag is set or not.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SAI_FLAG_OVRUDR: Overrun underrun flag.
  - SAI_FLAG_MUTEDET: Mute detection flag.
  - SAI_FLAG_WCKCFG: Wrong Clock Configuration flag.
  - SAI_FLAG_FREQ: FIFO request flag.
  - SAI_FLAG_CNRDY: Codec not ready flag.
  - SAI_FLAG_AFSDET: Anticipated frame synchronization detection flag.
  - SAI_FLAG_LFSDET: Late frame synchronization detection flag.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SAI_CLEAR_FLAG

Description:

- Clear the specified SAI pending flag.

Parameters:

- __HANDLE__: specifies the SAI Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - SAI_FLAG_OVRUDR: Clear Overrun underrun
  - SAI_FLAG_MUTEDET: Clear Mute detection
  - SAI_FLAG_WCKCFG: Clear Wrong Clock Configuration
  - SAI_FLAG_FREQ: Clear FIFO request
  - SAI_FLAG_CNRDY: Clear Codec not ready
  - SAI_FLAG_AFSDET: Clear Anticipated frame synchronization detection
  - SAI_FLAG_LFSDET: Clear Late frame synchronization detection

Return value:

- None

__HAL_SAI_ENABLE

Description:

- Enable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

__HAL_SAI_DISABLE

Description:

- Disable SAI.

Parameters:

- __HANDLE__: specifies the SAI Handle.

Return value:

- None

SAI Mono Stereo Mode

SAI_STEREOMODE

SAI_MONOMODE

SAI PDM Clock Enable

SAI_PDM_CLOCK1_ENABLE

SAI_PDM_CLOCK2_ENABLE

SAI Supported protocol

SAI_I2S_STANDARD

SAI_I2S_MSBJUSTIFIED

SAI_I2S_LSBJUSTIFIED

SAI_PCM_LONG

SAI_PCM_SHORT

SAI protocol data size

SAI_PROTOCOL_DATASIZE_16BIT

SAI_PROTOCOL_DATASIZE_16BITEXTENDED

SAI_PROTOCOL_DATASIZE_24BIT

SAI_PROTOCOL_DATASIZE_32BIT

SAI TRIState Management

SAI_OUTPUT_NOTRELEASED

SAI_OUTPUT_RELEASED

71 HAL SAI Extension Driver

71.1 SAIEx Firmware driver registers structures

71.1.1 SAIEx_PdmMicDelayParamTypeDef

SAIEx_PdmMicDelayParamTypeDef is defined in the stm32u5xx_hal_sai_ex.h

Data Fields

- ***uint32_t MicPair***
- ***uint32_t LeftDelay***
- ***uint32_t RightDelay***

Field Documentation

- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::MicPair***
  Specifies which pair of microphones is selected. This parameter must be a number between Min_Data = 1 and Max_Data = 3.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::LeftDelay***
  Specifies the delay in PDM clock unit to apply on left microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.
- ***uint32_t SAIEx_PdmMicDelayParamTypeDef::RightDelay***
  Specifies the delay in PDM clock unit to apply on right microphone. This parameter must be a number between Min_Data = 0 and Max_Data = 7.

71.2 SAIEx Firmware driver API description

The following section lists the various functions of the SAIEx library.

71.2.1 Extended features functions

This section provides functions allowing to:

- Modify PDM microphone delays

This section contains the following APIs:

- *HAL_SAIEx_ConfigPdmMicDelay()*

71.2.2 Detailed description of functions

HAL_SAIEx_ConfigPdmMicDelay

Function name

HAL_StatusTypeDef HAL_SAIEx_ConfigPdmMicDelay (const SAI_HandleTypeDef * hsai, const SAIEx_PdmMicDelayParamTypeDef * pdmMicDelay)

Function description

Configure PDM microphone delays.

Parameters

- **hsai:** SAI handle.
- **pdmMicDelay:** Microphone delays configuration.

Return values

- **HAL:** status

72 HAL SD Generic Driver

72.1 SD Firmware driver registers structures

72.1.1 HAL_SD_CardInfoTypeDef

HAL_SD_CardInfoTypeDef is defined in the stm32u5xx_hal_sd.h

Data Fields

- ***uint32_t CardType***
- ***uint32_t CardVersion***
- ***uint32_t Class***
- ***uint32_t RelCardAdd***
- ***uint32_t BlockNbr***
- ***uint32_t BlockSize***
- ***uint32_t LogBlockNbr***
- ***uint32_t LogBlockSize***
- ***uint32_t CardSpeed***

Field Documentation

- ***uint32_t HAL_SD_CardInfoTypeDef::CardType***
  Specifies the card Type
- ***uint32_t HAL_SD_CardInfoTypeDef::CardVersion***
  Specifies the card version
- ***uint32_t HAL_SD_CardInfoTypeDef::Class***
  Specifies the class of the card class
- ***uint32_t HAL_SD_CardInfoTypeDef::RelCardAdd***
  Specifies the Relative Card Address
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockNbr***
  Specifies the Card Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::BlockSize***
  Specifies one block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockNbr***
  Specifies the Card logical Capacity in blocks
- ***uint32_t HAL_SD_CardInfoTypeDef::LogBlockSize***
  Specifies logical block size in bytes
- ***uint32_t HAL_SD_CardInfoTypeDef::CardSpeed***
  Specifies the card Speed

72.1.2 __SD_HandleTypeDef

__SD_HandleTypeDef is defined in the stm32u5xx_hal_sd.h

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***const uint8_t * pTxBuffPtr***
- ***uint32_t TxXferSize***
- ***uint8_t * pRxBuffPtr***
- ***uint32_t RxXferSize***
- ***__IO uint32_t Context***
- ***__IO HAL_SD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***HAL_SD_CardInfoTypeDef SdCard***

- *uint32_t CSD*
- *uint32_t CID*
- *void(* TxCpltCallback*
- *void(* RxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* Read_DMALnkLstBufCpltCallback*
- *void(* Write_DMALnkLstBufCpltCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***SD_TypeDef* __SD_HandleTypeDef::Instance***
  SD registers base address
- ***SD_InitTypeDef __SD_HandleTypeDef::Init***
  SD required parameters
- ***HAL_LockTypeDef __SD_HandleTypeDef::Lock***
  SD locking object
- ***const uint8_t* __SD_HandleTypeDef::pTxBuffPtr***
  Pointer to SD Tx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::TxXferSize***
  SD Tx Transfer size
- ***uint8_t* __SD_HandleTypeDef::pRxBuffPtr***
  Pointer to SD Rx transfer Buffer
- ***uint32_t __SD_HandleTypeDef::RxXferSize***
  SD Rx Transfer size
- ***__IO uint32_t __SD_HandleTypeDef::Context***
  SD transfer context
- ***__IO HAL_SD_StateTypeDef __SD_HandleTypeDef::State***
  SD card State
- ***__IO uint32_t __SD_HandleTypeDef::ErrorCode***
  SD Card Error codes
- ***HAL_SD_CardInfoTypeDef __SD_HandleTypeDef::SdCard***
  SD Card information
- ***uint32_t __SD_HandleTypeDef::CSD[4]***
  SD card specific data table
- ***uint32_t __SD_HandleTypeDef::CID[4]***
  SD card identification number table
- ***void(* __SD_HandleTypeDef::TxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::RxCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::ErrorCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::AbortCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Read_DMALnkLstBufCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::Write_DMALnkLstBufCpltCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::MspInitCallback)(struct __SD_HandleTypeDef *hsd)***
- ***void(* __SD_HandleTypeDef::MspDeInitCallback)(struct __SD_HandleTypeDef *hsd)***

72.1.3 HAL_SD_CardCSDTypeDef

HAL_SD_CardCSDTypeDef is defined in the stm32u5xx_hal_sd.h

Data Fields

- *__IO uint8_t CSDStruct*
- *__IO uint8_t SysSpecVersion*
- *__IO uint8_t Reserved1*
- *__IO uint8_t TAAC*

- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGroup***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSDStruct***
  CSD structure
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::SysSpecVersion***
  System specification version
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved1***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TAAC***
  Data read access time 1
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::NSAC***
  Data read access time 2 in CLK cycles
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxBusClkFrec***
  Max. bus clock frequency
- ***__IO uint16_t HAL_SD_CardCSDTypeDef::CardComdClasses***
  Card command classes
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockLen***
  Max. read data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PartBlockRead***
  Partial blocks for read allowed

- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrBlockMisalign***
  Write block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::RdBlockMisalign***
  Read block misalignment
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DSRImpl***
  DSR implemented
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved2***
  Reserved
- ***__IO uint32_t HAL_SD_CardCSDTypeDef::DeviceSize***
  Device Size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMin***
  Max. read current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxRdCurrentVDDMax***
  Max. read current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMin***
  Max. write current @ VDD min
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrCurrentVDDMax***
  Max. write current @ VDD max
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::DeviceSizeMul***
  Device size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrSize***
  Erase group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::EraseGrMul***
  Erase group size multiplier
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrSize***
  Write protect group size
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrProtectGrEnable***
  Write protect group enable
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ManDeflECC***
  Manufacturer default ECC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WrSpeedFact***
  Write speed factor
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::MaxWrBlockLen***
  Max. write data block length
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::WriteBlockPaPartial***
  Partial blocks for write allowed
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved3***
  Reserved
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ContentProtectAppli***
  Content protection application
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormatGroup***
  File format group
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CopyFlag***
  Copy flag (OTP)
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::PermWrProtect***
  Permanent write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::TempWrProtect***
  Temporary write protection
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::FileFormat***
  File format
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::ECC***
  ECC code
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::CSD_CRC***
  CSD CRC
- ***__IO uint8_t HAL_SD_CardCSDTypeDef::Reserved4***
  Always 1

72.1.4 HAL_SD_CardCIDTypeDef

HAL_SD_CardCIDTypeDef is defined in the stm32u5xx_hal_sd.h

Data Fields

- ***__IO uint8_t ManufacturerID***
- ***__IO uint16_t OEM_AppliID***
- ***__IO uint32_t ProdName1***
- ***__IO uint8_t ProdName2***
- ***__IO uint8_t ProdRev***
- ***__IO uint32_t ProdSN***
- ***__IO uint8_t Reserved1***
- ***__IO uint16_t ManufactDate***
- ***__IO uint8_t CID_CRC***
- ***__IO uint8_t Reserved2***

Field Documentation

- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ManufacturerID***
  Manufacturer ID
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::OEM_AppliID***
  OEM/Application ID
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdName1***
  Product Name part1
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdName2***
  Product Name part2
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::ProdRev***
  Product Revision
- ***__IO uint32_t HAL_SD_CardCIDTypeDef::ProdSN***
  Product Serial Number
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved1***
  Reserved1
- ***__IO uint16_t HAL_SD_CardCIDTypeDef::ManufactDate***
  Manufacturing Date
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::CID_CRC***
  CID CRC
- ***__IO uint8_t HAL_SD_CardCIDTypeDef::Reserved2***
  Always 1

72.1.5 HAL_SD_CardStatusTypeDef

HAL_SD_CardStatusTypeDef is defined in the stm32u5xx_hal_sd.h

Data Fields

- ***__IO uint8_t DataBusWidth***
- ***__IO uint8_t SecuredMode***
- ***__IO uint16_t CardType***
- ***__IO uint32_t ProtectedAreaSize***
- ***__IO uint8_t SpeedClass***
- ***__IO uint8_t PerformanceMove***
- ***__IO uint8_t AllocationUnitSize***
- ***__IO uint16_t EraseSize***
- ***__IO uint8_t EraseTimeout***
- ***__IO uint8_t EraseOffset***
- ***__IO uint8_t UhsSpeedGrade***
- ***__IO uint8_t UhsAllocationUnitSize***
- ***__IO uint8_t VideoSpeedClass***

Field Documentation

- ***__IO uint8_t HAL_SD_CardStatusTypeDef::DataBusWidth***
  Shows the currently defined data bus width
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SecuredMode***
  Card is in secured mode of operation
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::CardType***
  Carries information about card type
- ***__IO uint32_t HAL_SD_CardStatusTypeDef::ProtectedAreaSize***
  Carries information about the capacity of protected area
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::SpeedClass***
  Carries information about the speed class of the card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::PerformanceMove***
  Carries information about the card's performance move
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::AllocationUnitSize***
  Carries information about the card's allocation unit size
- ***__IO uint16_t HAL_SD_CardStatusTypeDef::EraseSize***
  Determines the number of AUs to be erased in one operation
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseTimeout***
  Determines the timeout for any number of AU erase
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::EraseOffset***
  Carries information about the erase offset
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsSpeedGrade***
  Carries information about the speed grade of UHS card
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::UhsAllocationUnitSize***
  Carries information about the UHS card's allocation unit size
- ***__IO uint8_t HAL_SD_CardStatusTypeDef::VideoSpeedClass***
  Carries information about the Video Speed Class of UHS card

72.2 SD Firmware driver API description

The following section lists the various functions of the SD library.

72.2.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDMMC and GPIO) are performed by the user in HAL_SD_MspInit() function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDMMC memories which uses the HAL SDMMC driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDMMC low level resources by implementing the HAL_SD_MspInit() API:
   a. Enable the SDMMC interface clock using __HAL_RCC_SDMMC_CLK_ENABLE();
   b. SDMMC pins configuration for SD card
      - Enable the clock for the SDMMC GPIOs using the functions __HAL_RCC_GPIOx_CLK_ENABLE();
      - Configure these SDMMC pins as alternate function pull-up using HAL_GPIO_Init() and according to your pin assignment;
   c. NVIC configuration if you need to use interrupt process (HAL_SD_ReadBlocks_IT() and HAL_SD_WriteBlocks_IT() APIs).
      - Configure the SDMMC interrupt priorities using function HAL_NVIC_SetPriority();
      - Enable the NVIC SDMMC IRQs using function HAL_NVIC_EnableIRQ()
      - SDMMC interrupts are managed using the macros __HAL_SD_ENABLE_IT() and __HAL_SD_DISABLE_IT() inside the communication process.
      - SDMMC interrupts pending bits are managed using the macros __HAL_SD_GET_IT() and __HAL_SD_CLEAR_IT()
   d. No general propose DMA Configuration is needed, an Internal DMA for SDMMC Peripheral are used.
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

SD Card Initialization and configuration

To initialize the SD Card, use the HAL_SD_Init() function. It Initializes SDMMC Peripheral(STM32 side) and the SD Card, and put it into StandBy State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDMMC_CK) is computed as follows: SDMMC_CK = SDMMCCLK / (2 * ClockDiv) In initialization mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 400KHz. This phase of initialization is done through SDMMC_Init() and SDMMC_PowerState_ON() SDMMC low level APIs.
2. Initialize the SD card. The API used is HAL_SD_InitCard(). This phase allows the card initialization and identification and check the SD Card type (Standard Capacity or High Capacity) The initialization flow is compatible with SD standard. This API (HAL_SD_InitCard()) could be used also to reinitialize the card in case of plug-off plug-in.
3. Configure the SD Card Data transfer frequency. You can change or adapt this frequency by adjusting the "ClockDiv" field. In transfer mode and according to the SD Card standard, make sure that the SDMMC_CK frequency doesn't exceed 25MHz and 100MHz in High-speed mode switch.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function HAL_SD_ReadBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can read from SD card in DMA mode by using function HAL_SD_ReadBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Rx interrupt event.
- You can read from SD card in Interrupt mode by using function HAL_SD_ReadBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Rx interrupt event.

SD Card Write operation

- You can write to SD card in polling mode by using function HAL_SD_WriteBlocks(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state.
- You can write to SD card in DMA mode by using function HAL_SD_WriteBlocks_DMA(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the DMA transfer process through the SD Tx interrupt event.
- You can write to SD card in Interrupt mode by using function HAL_SD_WriteBlocks_IT(). This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to ensure that the transfer is done correctly. The check is done through HAL_SD_GetCardState() function for SD card state. You could also check the IT transfer process through the SD Tx interrupt event.

SD card status

- The SD Status contains status bits that are related to the SD Memory Card proprietary features. To get SD card status use the HAL_SD_GetCardStatus().

SD card information

- To get SD card information, you can use the function HAL_SD_GetCardInfo(). It returns useful information about the SD card such as block size, card type, block number ...

SD card CSD register

SD card CID register

SD HAL driver macros list

Note: *You can refer to the SD HAL driver header file for more useful macros*

Callback registration

The compilation define USE_HAL_SD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SD_RegisterCallback() to register a user callback, it allows to register following callbacks:

- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMADblBuf0CpltCallback : callback when the DMA reception of first buffer is completed.
- Read_DMADblBuf1CpltCallback : callback when the DMA reception of second buffer is completed.
- Write_DMADblBuf0CpltCallback : callback when the DMA transmission of first buffer is completed.
- Write_DMADblBuf1CpltCallback : callback when the DMA transmission of second buffer is completed.
- MspInitCallback : SD MspInit.
- MspDeInitCallback : SD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. For specific callbacks TransceiverCallback use dedicated register callbacks: respectively HAL_SD_RegisterTransceiverCallback(). Use function HAL_SD_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- TxCpltCallback : callback when a transmission transfer is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- Read_DMALnkLstBufCpltCallback : callback when the DMA reception of linked list node buffer is completed.
- Write_DMALnkLstBufCpltCallback : callback when the DMA transmission of linked list node buffer is completed.
- MspInitCallback : SD MspInit.

- MspDeInitCallback : SD MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. For specific callbacks TransceiverCallback use dedicated unregister callbacks: respectively HAL_SD_UnRegisterTransceiverCallback(). By default, after the HAL_SD_Init and if the state is HAL_SD_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SD_Init and HAL_SD_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SD_Init and HAL_SD_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/ unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/ unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SD_RegisterCallback before calling HAL_SD_DeInit or HAL_SD_Init function. When The compilation define USE_HAL_SD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

72.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- *HAL_SD_Init()*
- *HAL_SD_InitCard()*
- *HAL_SD_DeInit()*
- *HAL_SD_MspInit()*
- *HAL_SD_MspDeInit()*

72.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- *HAL_SD_ReadBlocks()*
- *HAL_SD_WriteBlocks()*
- *HAL_SD_ReadBlocks_IT()*
- *HAL_SD_WriteBlocks_IT()*
- *HAL_SD_ReadBlocks_DMA()*
- *HAL_SD_WriteBlocks_DMA()*
- *HAL_SD_Erase()*
- *HAL_SD_IRQHandler()*
- *HAL_SD_GetState()*
- *HAL_SD_GetError()*
- *HAL_SD_TxCpltCallback()*
- *HAL_SD_RxCpltCallback()*
- *HAL_SD_ErrorCallback()*
- *HAL_SD_AbortCallback()*
- *HAL_SD_RegisterCallback()*
- *HAL_SD_UnRegisterCallback()*

72.2.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations and get the related information

This section contains the following APIs:

- *HAL_SD_GetCardCID()*
- *HAL_SD_GetCardCSD()*
- *HAL_SD_GetCardStatus()*
- *HAL_SD_GetCardInfo()*

- *HAL_SD_ConfigWideBusOperation()*
- *HAL_SD_ConfigSpeedBusOperation()*
- *HAL_SD_GetCardState()*
- *HAL_SD_Abort()*
- *HAL_SD_Abort_IT()*

72.2.5 Detailed description of functions

HAL_SD_Init

Function name

HAL_StatusTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd)

Function description

Initializes the SD according to the specified parameters in the SD_HandleTypeDef and create the associated handle.

Parameters

- **hsd:** Pointer to the SD handle

Return values

- **HAL:** status

HAL_SD_InitCard

Function name

HAL_StatusTypeDef HAL_SD_InitCard (SD_HandleTypeDef * hsd)

Function description

Initializes the SD Card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

Notes

- This function initializes the SD card. It could be used when a card re-initialization is needed.

HAL_SD_DeInit

Function name

HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)

Function description

De-Initializes the SD card.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **HAL:** status

HAL_SD_MspInit

Function name

void HAL_SD_MspInit (SD_HandleTypeDef * hsd)

Function description

Initializes the SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_MspDeInit

Function name

void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)

Function description

De-Initialize SD MSP.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of SD blocks to read
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)

Function description

Allows to write block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of SD blocks to write
- **Timeout:** Specify timeout value

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_Erase

Function name

HAL_StatusTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint32_t BlockStartAdd, uint32_t BlockEndAdd)

Function description

Erases the specified memory area of the given SD card.

Parameters

- **hsd:** Pointer to SD handle
- **BlockStartAdd:** Start Block address
- **BlockEndAdd:** End Block address

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().

HAL_SD_ReadBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_IT (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_IT

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_IT (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the IT transfer process through the SD Tx interrupt event.

HAL_SD_ReadBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Reads block(s) from a specified address in a card.

Parameters

- **hsd:** Pointer SD handle
- **pData:** Pointer to the buffer that will contain the received data
- **BlockAdd:** Block Address from where data is to be read
- **NumberOfBlocks:** Number of blocks to read.

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Rx interrupt event.

HAL_SD_WriteBlocks_DMA

Function name

HAL_StatusTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, const uint8_t * pData, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

Writes block(s) to a specified address in a card.

Parameters

- **hsd:** Pointer to SD handle
- **pData:** Pointer to the buffer that will contain the data to transmit
- **BlockAdd:** Block Address where data will be written
- **NumberOfBlocks:** Number of blocks to write

Return values

- **HAL:** status

Notes

- This API should be followed by a check on the card state through HAL_SD_GetCardState().
- You could also check the DMA transfer process through the SD Tx interrupt event.

HAL_SD_IRQHandler

Function name

void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)

Function description

This function handles SD card interrupt request.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_TxCpltCallback

Function name

void HAL_SD_TxCpltCallback (SD_HandleTypeDef * hsd)

Function description

Tx Transfer completed callbacks.

Parameters

- **hsd:** Pointer to SD handle

Return values

- **None:**

HAL_SD_RxCpltCallback

Function name

void HAL_SD_RxCpltCallback (SD_HandleTypeDef * hsd)

Function description

Rx Transfer completed callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_ErrorCallback

Function name

void HAL_SD_ErrorCallback (SD_HandleTypeDef * hsd)

Function description

SD error callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_AbortCallback

Function name

void HAL_SD_AbortCallback (SD_HandleTypeDef * hsd)

Function description

SD Abort callbacks.

Parameters

- **hsd:** Pointer SD handle

Return values

- **None:**

HAL_SD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_RegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID, pSD_CallbackTypeDef pCallback)

Function description

Register a User SD Callback To be used instead of the weak (overridden) predefined callback.

Parameters

- **hsd:** : SD handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  – HAL_SD_TX_CPLT_CB_ID SD Tx Complete Callback ID
  – HAL_SD_RX_CPLT_CB_ID SD Rx Complete Callback ID
  – HAL_SD_ERROR_CB_ID SD Error Callback ID
  – HAL_SD_ABORT_CB_ID SD Abort Callback ID
  – HAL_SD_READ_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Rx Linked List Node buffer Callback ID
  – HAL_SD_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Tx Linked List Node buffer Callback ID
  – HAL_SD_MSP_INIT_CB_ID SD MspInit Callback ID
  – HAL_SD_MSP_DEINIT_CB_ID SD MspDeInit Callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

Notes

- The HAL_SD_RegisterCallback() may be called before HAL_SD_Init() in HAL_SD_STATE_RESET to register callbacks for HAL_SD_MSP_INIT_CB_ID and HAL_SD_MSP_DEINIT_CB_ID.

HAL_SD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SD_UnRegisterCallback (SD_HandleTypeDef * hsd, HAL_SD_CallbackIDTypeDef CallbackID)

Function description

Unregister a User SD Callback SD Callback is redirected to the weak (overridden) predefined callback.

Parameters

- **hsd:** : SD handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_SD_TX_CPLT_CB_ID SD Tx Complete Callback ID
  – HAL_SD_RX_CPLT_CB_ID SD Rx Complete Callback ID
  – HAL_SD_ERROR_CB_ID SD Error Callback ID
  – HAL_SD_ABORT_CB_ID SD Abort Callback ID
  – HAL_SD_READ_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Rx Linked List Node buffer Callback ID
  – HAL_SD_WRITE_DMA_LNKLST_BUF_CPLT_CB_ID SD DMA Tx Linked List Node buffer Callback ID
  – HAL_SD_MSP_INIT_CB_ID SD MspInit Callback ID
  – HAL_SD_MSP_DEINIT_CB_ID SD MspDeInit Callback ID

Return values

- **status:**

Notes

- The HAL_SD_UnRegisterCallback() may be called before HAL_SD_Init() in HAL_SD_STATE_RESET to register callbacks for HAL_SD_MSP_INIT_CB_ID and HAL_SD_MSP_DEINIT_CB_ID.

HAL_SD_ConfigWideBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigWideBusOperation (SD_HandleTypeDef * hsd, uint32_t WideMode)

Function description

Enables wide bus operation for the requested card if supported by card.

Parameters

- **hsd:** Pointer to SD handle
- **WideMode:** Specifies the SD card wide bus mode This parameter can be one of the following values:
  – SDMMC_BUS_WIDE_8B: 8-bit data transfer
  – SDMMC_BUS_WIDE_4B: 4-bit data transfer
  – SDMMC_BUS_WIDE_1B: 1-bit data transfer

Return values

- **HAL:** status

HAL_SD_ConfigSpeedBusOperation

Function name

HAL_StatusTypeDef HAL_SD_ConfigSpeedBusOperation (SD_HandleTypeDef * hsd, uint32_t SpeedMode)

Function description

Configure the speed bus mode.

Parameters

- **hsd:** Pointer to the SD handle
- **SpeedMode:** Specifies the SD card speed bus mode This parameter can be one of the following values:
  - SDMMC_SPEED_MODE_AUTO: Max speed mode supported by the card
  - SDMMC_SPEED_MODE_DEFAULT: Default Speed/SDR12 mode
  - SDMMC_SPEED_MODE_HIGH: High Speed/SDR25 mode
  - SDMMC_SPEED_MODE_ULTRA: Ultra high speed mode

Return values

- **HAL:** status

HAL_SD_GetCardState

Function name

HAL_SD_CardStateTypeDef HAL_SD_GetCardState (SD_HandleTypeDef * hsd)

Function description

Gets the current sd card data state.

Parameters

- **hsd:** pointer to SD handle

Return values

- **Card:** state

HAL_SD_GetCardCID

Function name

HAL_StatusTypeDef HAL_SD_GetCardCID (SD_HandleTypeDef * hsd, HAL_SD_CardCIDTypeDef * pCID)

Function description

Returns information the information of the card which are stored on the CID register.

Parameters

- **hsd:** Pointer to SD handle
- **pCID:** Pointer to a HAL_SD_CardCIDTypeDef structure that contains all CID register parameters

Return values

- **HAL:** status

HAL_SD_GetCardCSD

Function name

HAL_StatusTypeDef HAL_SD_GetCardCSD (SD_HandleTypeDef * hsd, HAL_SD_CardCSDTypeDef * pCSD)

Function description

Returns information the information of the card which are stored on the CSD register.

Parameters

- **hsd:** Pointer to SD handle
- **pCSD:** Pointer to a HAL_SD_CardCSDTypeDef structure that contains all CSD register parameters

Return values

- **HAL:** status

HAL_SD_GetCardStatus

Function name

HAL_StatusTypeDef HAL_SD_GetCardStatus (SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pStatus)

Function description

Gets the SD status info.

Parameters

- **hsd:** Pointer to SD handle
- **pStatus:** Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetCardInfo

Function name

HAL_StatusTypeDef HAL_SD_GetCardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)

Function description

Gets the SD card info.

Parameters

- **hsd:** Pointer to SD handle
- **pCardInfo:** Pointer to the HAL_SD_CardInfoTypeDef structure that will contain the SD card status information

Return values

- **HAL:** status

HAL_SD_GetState

Function name

HAL_SD_StateTypeDef HAL_SD_GetState (const SD_HandleTypeDef * hsd)

Function description

return the SD state

Parameters

- **hsd:** Pointer to sd handle

Return values

- **HAL:** state

HAL_SD_GetError

Function name

uint32_t HAL_SD_GetError (const SD_HandleTypeDef * hsd)

Function description

Return the SD error code.

Parameters

- **hsd:** : Pointer to a SD_HandleTypeDef structure that contains the configuration information.

Return values

- **SD:** Error Code

HAL_SD_Abort

Function name

HAL_StatusTypeDef HAL_SD_Abort (SD_HandleTypeDef * hsd)

Function description

Abort the current transfer and disable the SD.

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

HAL_SD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SD_Abort_IT (SD_HandleTypeDef * hsd)

Function description

Abort the current transfer and disable the SD (IT mode).

Parameters

- **hsd:** pointer to a SD_HandleTypeDef structure that contains the configuration information for SD module.

Return values

- **HAL:** status

72.3 SD Firmware driver defines

The following section lists the various define and macros of the module.

72.3.1 SD

SD

SD Error status enumeration Structure definition

HAL_SD_ERROR_NONE

No error

HAL_SD_ERROR_CMD_CRC_FAIL

Command response received (but CRC check failed)

HAL_SD_ERROR_DATA_CRC_FAIL

Data block sent/received (CRC check failed)

HAL_SD_ERROR_CMD_RSP_TIMEOUT

Command response timeout

HAL_SD_ERROR_DATA_TIMEOUT

Data timeout

HAL_SD_ERROR_TX_UNDERRUN

Transmit FIFO underrun

HAL_SD_ERROR_RX_OVERRUN

Receive FIFO overrun

HAL_SD_ERROR_ADDR_MISALIGNED

Misaligned address

HAL_SD_ERROR_BLOCK_LEN_ERR

Transferred block length is not allowed for the card or the number of transferred bytes does not match the block length

HAL_SD_ERROR_ERASE_SEQ_ERR

An error in the sequence of erase command occurs

HAL_SD_ERROR_BAD_ERASE_PARAM

An invalid selection for erase groups

HAL_SD_ERROR_WRITE_PROT_VIOLATION

Attempt to program a write protect block

HAL_SD_ERROR_LOCK_UNLOCK_FAILED

Sequence or password error has been detected in unlock command or if there was an attempt to access a locked card

HAL_SD_ERROR_COM_CRC_FAILED

CRC check of the previous command failed

HAL_SD_ERROR_ILLEGAL_CMD

Command is not legal for the card state

HAL_SD_ERROR_CARD_ECC_FAILED

Card internal ECC was applied but failed to correct the data

HAL_SD_ERROR_CC_ERR

Internal card controller error

HAL_SD_ERROR_GENERAL_UNKNOWN_ERR

General or unknown error

HAL_SD_ERROR_STREAM_READ_UNDERRUN

The card could not sustain data reading in stream rmode

HAL_SD_ERROR_STREAM_WRITE_OVERRUN

The card could not sustain data programming in stream mode

HAL_SD_ERROR_CID_CSD_OVERWRITE

CID/CSD overwrite error

HAL_SD_ERROR_WP_ERASE_SKIP
Only partial address space was erased

HAL_SD_ERROR_CARD_ECC_DISABLED
Command has been executed without using internal ECC

HAL_SD_ERROR_ERASE_RESET
Erase sequence was cleared before executing because an out of erase sequence command was received

HAL_SD_ERROR_AKE_SEQ_ERR
Error in sequence of authentication

HAL_SD_ERROR_INVALID_VOLTRANGE
Error in case of invalid voltage range

HAL_SD_ERROR_ADDR_OUT_OF_RANGE
Error when addressed block is out of range

HAL_SD_ERROR_REQUEST_NOT_APPLICABLE
Error when command request is not applicable

HAL_SD_ERROR_PARAM
the used parameter is not valid

HAL_SD_ERROR_UNSUPPORTED_FEATURE
Error when feature is not insupported

HAL_SD_ERROR_BUSY
Error when transfer process is busy

HAL_SD_ERROR_DMA
Error while DMA transfer

HAL_SD_ERROR_TIMEOUT
Timeout error

HAL_SD_ERROR_INVALID_CALLBACK
Invalid callback error

SD context enumeration

SD_CONTEXT_NONE
None

SD_CONTEXT_READ_SINGLE_BLOCK
Read single block operation

SD_CONTEXT_READ_MULTIPLE_BLOCK
Read multiple blocks operation

SD_CONTEXT_WRITE_SINGLE_BLOCK
Write single block operation

SD_CONTEXT_WRITE_MULTIPLE_BLOCK
Write multiple blocks operation

SD_CONTEXT_IT
Process in Interrupt mode

SD_CONTEXT_DMA

Process in DMA mode

SD Supported Memory Cards

CARD_NORMAL_SPEED

Normal Speed Card <12.5Mo/s , Spec Version 1.01

CARD_HIGH_SPEED

High Speed Card <25Mo/s , Spec version 2.00

CARD_ULTRA_HIGH_SPEED

UHS-I SD Card <50Mo/s for SDR50, DDR5 Cards and <104Mo/s for SDR104, Spec version 3.01

CARD_SDSC

SD Standard Capacity <2Go

CARD_SDHC_SDXC

SD High Capacity <32Go, SD Extended Capacity <2To

CARD_SECURED

SD Supported Version

CARD_V1_X

CARD_V2_X

SD Exported Constants

BLOCKSIZE

Block size is 512 bytes

SD Exported Macros

__HAL_SD_RESET_HANDLE_STATE

Description:
- Reset SD handle state.

Parameters:
- __HANDLE__: SD Handle.

Return value:
- None

__HAL_SD_ENABLE_IT

Description:

- Enable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be enabled. This parameter can be one or a combination of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_DISABLE_IT

Description:

- Disable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt sources to be disabled. This parameter can be one or a combination of the following values:
  – SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  – SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  – SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  – SDMMC_IT_DTIMEOUT: Data timeout interrupt
  – SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  – SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  – SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  – SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  – SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  – SDMMC_IT_DHOLD: Data transfer Hold interrupt
  – SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  – SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  – SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  – SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  – SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  – SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  – SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  – SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  – SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  – SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  – SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  – SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  – SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

__HAL_SD_GET_FLAG

Description:

- Check whether the specified SD flag is set or not.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  - SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  - SDMMC_FLAG_CTIMEOUT: Command response timeout
  - SDMMC_FLAG_DTIMEOUT: Data timeout
  - SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  - SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  - SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  - SDMMC_FLAG_CMDSENT: Command sent (no response required)
  - SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  - SDMMC_FLAG_DHOLD: Data transfer Hold
  - SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  - SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  - SDMMC_FLAG_DPSMACT: Data path state machine active
  - SDMMC_FLAG_CPSMACT: Command path state machine active
  - SDMMC_FLAG_TXFIFOHE: Transmit FIFO Half Empty
  - SDMMC_FLAG_RXFIFOHF: Receive FIFO Half Full
  - SDMMC_FLAG_TXFIFOF: Transmit FIFO full
  - SDMMC_FLAG_RXFIFOF: Receive FIFO full
  - SDMMC_FLAG_TXFIFOE: Transmit FIFO empty
  - SDMMC_FLAG_RXFIFOE: Receive FIFO empty
  - SDMMC_FLAG_BUSYD0: Inverted value of SDMMC_D0 line (Busy)
  - SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  - SDMMC_FLAG_SDIOIT: SDIO interrupt received
  - SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  - SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  - SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  - SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  - SDMMC_FLAG_IDMATE: IDMA transfer error
  - SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- The: new state of SD FLAG (SET or RESET).

__HAL_SD_CLEAR_FLAG

Description:

- Clear the SD's pending flags.

Parameters:

- __HANDLE__: SD Handle.
- __FLAG__: specifies the flag to clear. This parameter can be one or a combination of the following values:
  – SDMMC_FLAG_CCRCFAIL: Command response received (CRC check failed)
  – SDMMC_FLAG_DCRCFAIL: Data block sent/received (CRC check failed)
  – SDMMC_FLAG_CTIMEOUT: Command response timeout
  – SDMMC_FLAG_DTIMEOUT: Data timeout
  – SDMMC_FLAG_TXUNDERR: Transmit FIFO underrun error
  – SDMMC_FLAG_RXOVERR: Received FIFO overrun error
  – SDMMC_FLAG_CMDREND: Command response received (CRC check passed)
  – SDMMC_FLAG_CMDSENT: Command sent (no response required)
  – SDMMC_FLAG_DATAEND: Data end (data counter, DATACOUNT, is zero)
  – SDMMC_FLAG_DHOLD: Data transfer Hold
  – SDMMC_FLAG_DBCKEND: Data block sent/received (CRC check passed)
  – SDMMC_FLAG_DABORT: Data transfer aborted by CMD12
  – SDMMC_FLAG_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected
  – SDMMC_FLAG_SDIOIT: SDIO interrupt received
  – SDMMC_FLAG_ACKFAIL: Boot Acknowledgment received
  – SDMMC_FLAG_ACKTIMEOUT: Boot Acknowledgment timeout
  – SDMMC_FLAG_VSWEND: Voltage switch critical timing section completion
  – SDMMC_FLAG_CKSTOP: SDMMC_CK stopped in Voltage switch procedure
  – SDMMC_FLAG_IDMATE: IDMA transfer error
  – SDMMC_FLAG_IDMABTC: IDMA buffer transfer complete

Return value:

- None

__HAL_SD_GET_IT

Description:

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the SDMMC interrupt source to check. This parameter can be one of the following values:
  - SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  - SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  - SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  - SDMMC_IT_DTIMEOUT: Data timeout interrupt
  - SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  - SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  - SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  - SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  - SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  - SDMMC_IT_DHOLD: Data transfer Hold interrupt
  - SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  - SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  - SDMMC_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
  - SDMMC_IT_RXFIFOHF: Receive FIFO Half Full interrupt
  - SDMMC_IT_RXFIFOF: Receive FIFO full interrupt
  - SDMMC_IT_TXFIFOE: Transmit FIFO empty interrupt
  - SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  - SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  - SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  - SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  - SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  - SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  - SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- The: new state of SD IT (SET or RESET).

__HAL_SD_CLEAR_IT

Description:

- Clear the SD's interrupt pending bits.

Parameters:

- __HANDLE__: SD Handle.
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
  – SDMMC_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
  – SDMMC_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
  – SDMMC_IT_CTIMEOUT: Command response timeout interrupt
  – SDMMC_IT_DTIMEOUT: Data timeout interrupt
  – SDMMC_IT_TXUNDERR: Transmit FIFO underrun error interrupt
  – SDMMC_IT_RXOVERR: Received FIFO overrun error interrupt
  – SDMMC_IT_CMDREND: Command response received (CRC check passed) interrupt
  – SDMMC_IT_CMDSENT: Command sent (no response required) interrupt
  – SDMMC_IT_DATAEND: Data end (data counter, DATACOUNT, is zero) interrupt
  – SDMMC_IT_DHOLD: Data transfer Hold interrupt
  – SDMMC_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
  – SDMMC_IT_DABORT: Data transfer aborted by CMD12 interrupt
  – SDMMC_IT_BUSYD0END: End of SDMMC_D0 Busy following a CMD response detected interrupt
  – SDMMC_IT_SDIOIT: SDIO interrupt received interrupt
  – SDMMC_IT_ACKFAIL: Boot Acknowledgment received interrupt
  – SDMMC_IT_ACKTIMEOUT: Boot Acknowledgment timeout interrupt
  – SDMMC_IT_VSWEND: Voltage switch critical timing section completion interrupt
  – SDMMC_IT_CKSTOP: SDMMC_CK stopped in Voltage switch procedure interrupt
  – SDMMC_IT_IDMABTC: IDMA buffer transfer complete interrupt

Return value:

- None

SD Card State enumeration structure

HAL_SD_CARD_READY

Card state is ready

HAL_SD_CARD_IDENTIFICATION

Card is in identification state

HAL_SD_CARD_STANDBY

Card is in standby state

HAL_SD_CARD_TRANSFER

Card is in transfer state

HAL_SD_CARD_SENDING

Card is sending an operation

HAL_SD_CARD_RECEIVING

Card is receiving operation information

HAL_SD_CARD_PROGRAMMING

Card is in programming state

HAL_SD_CARD_DISCONNECTED

Card is disconnected

HAL_SD_CARD_ERROR

Card response Error

SD Handle Structure definition

SD_InitTypeDef

SD_TypeDef

73 HAL SD Extension Driver

73.1 SDEx Firmware driver API description

The following section lists the various functions of the SDEx library.

73.1.1 How to use this driver

The SD Extension HAL driver can be used as follows:

- Configure Buffer0 and Buffer1 start address and Buffer size using HAL_SDEx_ConfigDMAMultiBuffer() function.
- Start Read and Write for multibuffer mode using HAL_SDEx_ReadBlocksDMAMultiBuffer() and HAL_SDEx_WriteBlocksDMAMultiBuffer() functions.

73.1.2 Linked List management functions

This subsection provides a set of functions allowing to manage the needed functions.

This section contains the following APIs:

- *HAL_SDEx_DMALinkedList_BuildNode()*
- *HAL_SDEx_DMALinkedList_InsertNode()*
- *HAL_SDEx_DMALinkedList_RemoveNode()*
- *HAL_SDEx_DMALinkedList_LockNode()*
- *HAL_SDEx_DMALinkedList_UnlockNode()*
- *HAL_SDEx_DMALinkedList_EnableCircularMode()*
- *HAL_SDEx_DMALinkedList_DisableCircularMode()*
- *HAL_SDEx_DMALinkedList_ReadBlocks()*
- *HAL_SDEx_DMALinkedList_WriteBlocks()*
- *HAL_SDEx_DMALinkedList_ReadBlocks()*
- *HAL_SDEx_DMALinkedList_WriteBlocks()*
- *HAL_SDEx_Read_DMALnkLstBufCpltCallback()*
- *HAL_SDEx_Write_DMALnkLstBufCpltCallback()*

73.1.3 Detailed description of functions

HAL_SDEx_DMALinkedList_ReadBlocks

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_ReadBlocks (SD_HandleTypeDef * hsd, SD_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SDEx_DMALinkedList_WriteBlocks

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_WriteBlocks (SD_HandleTypeDef * hsd, SD_DMALinkedListTypeDef * pLinkedList, uint32_t BlockAdd, uint32_t NumberOfBlocks)

Function description

HAL_SDEx_DMALinkedList_BuildNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_BuildNode (SD_DMALinkNodeTypeDef * pNode, SD_DMALinkNodeConfTypeDef * pNodeConf)

Function description

Build Linked List node.

Parameters

- **pNode:** Pointer to new node to add.
- **pNodeConf:** Pointer to configuration parameters for new node to add.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_InsertNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_InsertNode (SD_DMALinkedListTypeDef * pLinkedList, SD_DMALinkNodeTypeDef * pPrevNode, SD_DMALinkNodeTypeDef * pNewNode)

Function description

Insert new Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pPrevNode:** Pointer to previous node.
- **pNewNode:** Pointer to new node to insert.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_RemoveNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_RemoveNode (SD_DMALinkedListTypeDef * pLinkedList, SD_DMALinkNodeTypeDef * pNode)

Function description

Remove Linked List node.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes
- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_LockNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_LockNode (SD_DMALinkNodeTypeDef * pNode)

Function description

Lock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_UnlockNode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_UnlockNode (SD_DMALinkNodeTypeDef * pNode)

Function description

Unlock Linked List node.

Parameters

- **pNode:** Pointer to node to remove.

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_EnableCircularMode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_EnableCircularMode (SD_DMALinkedListTypeDef * pLinkedList)

Function description

Enable Circular mode for DMA Linked List.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes

Return values

- **HAL:** status

HAL_SDEx_DMALinkedList_DisableCircularMode

Function name

HAL_StatusTypeDef HAL_SDEx_DMALinkedList_DisableCircularMode (SD_DMALinkedListTypeDef * pLinkedList)

Function description

Disable Circular mode for DMA Linked List.

Parameters

- **pLinkedList:** Pointer to the linkedlist that contains transfer nodes

Return values

- **HAL:** status

HAL_SDEx_Read_DMALnkLstBufCpltCallback

Function name

void HAL_SDEx_Read_DMALnkLstBufCpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

HAL_SDEx_Write_DMALnkLstBufCpltCallback

Function name

void HAL_SDEx_Write_DMALnkLstBufCpltCallback (SD_HandleTypeDef * hsd)

Function description

Read DMA Linked list node Transfer completed callbacks.

Parameters

- **hsd:** SD handle

Return values

- **None:**

73.2 SDEx Firmware driver defines

The following section lists the various define and macros of the module.

73.2.1 SDEx

SDEx

Linked List Wrapper

SD_DMALinkNodeTypeDef

SD_DMALinkNodeConfTypeDef

SD_DMALinkedListTypeDef

HAL_SDEx_DMALinkedList_WriteCpltCallback

HAL_SDEx_DMALinkedList_ReadCpltCallback

74 HAL SMARTCARD Generic Driver

74.1 SMARTCARD Firmware driver registers structures

74.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32u5xx_hal_smartcard.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint16_t Parity***
- ***uint16_t Mode***
- ***uint16_t CLKPolarity***
- ***uint16_t CLKPhase***
- ***uint16_t CLKLastBit***
- ***uint16_t OneBitSampling***
- ***uint8_t Prescaler***
- ***uint8_t GuardTime***
- ***uint16_t NACKEnable***
- ***uint32_t TimeOutEnable***
- ***uint32_t TimeOutValue***
- ***uint8_t BlockLength***
- ***uint8_t AutoRetryCount***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
  Configures the SmartCard communication baud rate. The baud rate register is computed using the following formula: Baud Rate Register = ((usart_ker_ckpres) / ((hsmartcard->Init.BaudRate))) where usart_ker_ckpres is the USART input clock divided by a prescaler
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter ***SMARTCARD_Word_Length*** can only be set to 9 (8 data + 1 parity bits).
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
  Specifies the number of stop bits. This parameter can be a value of ***SMARTCARD_Stop_Bits***.
- ***uint16_t SMARTCARD_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***SMARTCARD_Parity***
  Note:
  – The parity is enabled by default (PCE is forced to 1). Since the WordLength is forced to 8 bits + parity, M is forced to 1 and the parity bit is the 9th bit.
- ***uint16_t SMARTCARD_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***SMARTCARD_Mode***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***SMARTCARD_Clock_Polarity***
- ***uint16_t SMARTCARD_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***SMARTCARD_Clock_Phase***
- ***uint16_t SMARTCARD_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***SMARTCARD_Last_Bit***

- ***uint16_t SMARTCARD_InitTypeDef::OneBitSampling***
Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***SMARTCARD_OneBit_Sampling***.
- ***uint8_t SMARTCARD_InitTypeDef::Prescaler***
Specifies the SmartCard Prescaler. This parameter can be any value from 0x01 to 0x1F. Prescaler value is multiplied by 2 to give the division factor of the source clock frequency
- ***uint8_t SMARTCARD_InitTypeDef::GuardTime***
Specifies the SmartCard Guard Time applied after stop bits.
- ***uint16_t SMARTCARD_InitTypeDef::NACKEnable***
Specifies whether the SmartCard NACK transmission is enabled in case of parity error. This parameter can be a value of ***SMARTCARD_NACK_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutEnable***
Specifies whether the receiver timeout is enabled. This parameter can be a value of ***SMARTCARD_Timeout_Enable***
- ***uint32_t SMARTCARD_InitTypeDef::TimeOutValue***
Specifies the receiver time out value in number of baud blocks: it is used to implement the Character Wait Time (CWT) and Block Wait Time (BWT). It is coded over 24 bits.
- ***uint8_t SMARTCARD_InitTypeDef::BlockLength***
Specifies the SmartCard Block Length in T=1 Reception mode. This parameter can be any value from 0x0 to 0xFF
- ***uint8_t SMARTCARD_InitTypeDef::AutoRetryCount***
Specifies the SmartCard auto-retry count (number of retries in receive and transmit mode). When set to 0, retransmission is disabled. Otherwise, its maximum value is 7 (before signalling an error)
- ***uint32_t SMARTCARD_InitTypeDef::ClockPrescaler***
Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***SMARTCARD_ClockPrescaler***.

74.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32u5xx_hal_smartcard.h

Data Fields

- ***uint32_t AdvFeatureInit***
- ***uint32_t TxPinLevelInvert***
- ***uint32_t RxPinLevelInvert***
- ***uint32_t DataInvert***
- ***uint32_t Swap***
- ***uint32_t OverrunDisable***
- ***uint32_t DMADisableonRxError***
- ***uint32_t MSBFirst***
- ***uint16_t TxCompletionIndication***

Field Documentation

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::AdvFeatureInit***
Specifies which advanced SMARTCARD features is initialized. Several advanced features may be initialized at the same time. This parameter can be a value of ***SMARTCARDEx_Advanced_Features_Initialization_Type***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::TxPinLevelInvert***
Specifies whether the TX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Tx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::RxPinLevelInvert***
Specifies whether the RX pin active level is inverted. This parameter can be a value of ***SMARTCARD_Rx_Inv***
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DataInvert***
Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of ***SMARTCARD_Data_Inv***

- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::Swap***
  Specifies whether TX and RX pins are swapped. This parameter can be a value of *SMARTCARD_Rx_Tx_Swap*
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::OverrunDisable***
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of *SMARTCARD_Overrun_Disable*
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::DMADisableonRxError***
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of *SMARTCARD_DMA_Disable_on_Rx_Error*
- ***uint32_t SMARTCARD_AdvFeatureInitTypeDef::MSBFirst***
  Specifies whether MSB is sent first on UART line. This parameter can be a value of *SMARTCARD_MSB_First*
- ***uint16_t SMARTCARD_AdvFeatureInitTypeDef::TxCompletionIndication***
  Specifies which transmission completion indication is used: before (when relevant flag is available) or once guard time period has elapsed. This parameter can be a value of *SMARTCARDEx_Transmission_Completion_Indication*.

74.1.3 __SMARTCARD_HandleTypeDef

__SMARTCARD_HandleTypeDef is defined in the stm32u5xx_hal_smartcard.h

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***SMARTCARD_AdvFeatureInitTypeDef AdvancedInit***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***uint16_t NbRxDataToProcess***
- ***uint16_t NbTxDataToProcess***
- ***uint32_t FifoMode***
- ***void(* RxISR***
- ***void(* TxISR***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef gState***
- ***__IO HAL_SMARTCARD_StateTypeDef RxState***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* AbortTransmitCpltCallback***
- ***void(* AbortReceiveCpltCallback***
- ***void(* RxFifoFullCallback***
- ***void(* TxFifoEmptyCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***USART_TypeDef* __SMARTCARD_HandleTypeDef::Instance***
  USART registers base address
- ***SMARTCARD_InitTypeDef __SMARTCARD_HandleTypeDef::Init***
  SmartCard communication parameters
- ***SMARTCARD_AdvFeatureInitTypeDef __SMARTCARD_HandleTypeDef::AdvancedInit***
  SmartCard advanced features initialization parameters
- ***const uint8_t* __SMARTCARD_HandleTypeDef::pTxBuffPtr***
  Pointer to SmartCard Tx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::TxXferSize***
  SmartCard Tx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::TxXferCount***
  SmartCard Tx Transfer Counter
- ***uint8_t* __SMARTCARD_HandleTypeDef::pRxBuffPtr***
  Pointer to SmartCard Rx transfer Buffer
- ***uint16_t __SMARTCARD_HandleTypeDef::RxXferSize***
  SmartCard Rx Transfer size
- ***__IO uint16_t __SMARTCARD_HandleTypeDef::RxXferCount***
  SmartCard Rx Transfer Counter
- ***uint16_t __SMARTCARD_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution
- ***uint16_t __SMARTCARD_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __SMARTCARD_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***SMARTCARDEx_FIFO_mode***.
- ***void(* __SMARTCARD_HandleTypeDef::RxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __SMARTCARD_HandleTypeDef::TxISR)(struct __SMARTCARD_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmatx***
  SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SMARTCARD_HandleTypeDef::hdmarx***
  SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef __SMARTCARD_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::gState***
  SmartCard state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO HAL_SMARTCARD_StateTypeDef __SMARTCARD_HandleTypeDef::RxState***
  SmartCard state information related to Rx operations. This parameter can be a value of **HAL_SMARTCARD_StateTypeDef**
- ***__IO uint32_t __SMARTCARD_HandleTypeDef::ErrorCode***
  SmartCard Error code
- ***void(* __SMARTCARD_HandleTypeDef::TxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::ErrorCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Error Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Complete Callback

- ***void(* __SMARTCARD_HandleTypeDef::AbortTransmitCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Transmit Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::AbortReceiveCpltCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Abort Receive Complete Callback
- ***void(* __SMARTCARD_HandleTypeDef::RxFifoFullCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Rx Fifo Full Callback
- ***void(* __SMARTCARD_HandleTypeDef::TxFifoEmptyCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Tx Fifo Empty Callback
- ***void(* __SMARTCARD_HandleTypeDef::MspInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp Init callback
- ***void(* __SMARTCARD_HandleTypeDef::MspDeInitCallback)(struct __SMARTCARD_HandleTypeDef *hsmartcard)***
  SMARTCARD Msp DeInit callback

74.2 SMARTCARD Firmware driver API description

The following section lists the various functions of the SMARTCARD library.

74.2.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure (eg. SMARTCARD_HandleTypeDef hsmartcard).
2. Associate a USART to the SMARTCARD handle hsmartcard.
3. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
   – Enable the USARTx interface clock.
   – USART pins configuration:
     ◦ Enable the clock for the USART GPIOs.
     ◦ Configure the USART pins (TX as alternate function pull-up, RX as alternate function Input).
   – NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
     ◦ Configure the USARTx interrupt priority.
     ◦ Enable the NVIC USART IRQ handle.
   – DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
     ◦ Declare a DMA handle structure for the Tx/Rx channel.
     ◦ Enable the DMAx interface clock.
     ◦ Configure the declared DMA handle structure with the required Tx/Rx parameters.
     ◦ Configure the DMA Tx/Rx channel.
     ◦ Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
4. Program the Baud Rate, Parity, Mode(Receiver/Transmitter), clock enabling/disabling and accordingly, the clock parameters (parity, phase, last bit), prescaler value, guard time and NACK on transmission error enabling or disabling in the hsmartcard handle Init structure.
5. If required, program SMARTCARD advanced features (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard handle AdvancedInit structure.
6. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit() API.

Note: The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non-blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

DMA mode IO operation

- Send an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback()
- Receive an amount of data in non-blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback() is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback()
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback()

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_GET_FLAG : Check whether or not the specified SMARTCARD flag is set
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether or not the specified SMARTCARD interrupt is enabled

Note: You can refer to the SMARTCARD HAL driver header file for more useful macros

74.2.2 Callback registration

The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_SMARTCARD_RegisterCallback() to register a user callback. Function HAL_SMARTCARD_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.

- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_SMARTCARD_UnRegisterCallback() to reset a callback to the default weak function. HAL_SMARTCARD_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxCpltCallback : Tx Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : SMARTCARD MspInit.
- MspDeInitCallback : SMARTCARD MspDeInit.

By default, after the HAL_SMARTCARD_Init() and when the state is HAL_SMARTCARD_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SMARTCARD_Init() and HAL_SMARTCARD_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_SMARTCARD_STATE_READY state only. Exception done MspInit/MspDeInit that can be registered/unregistered in HAL_SMARTCARD_STATE_READY or HAL_SMARTCARD_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SMARTCARD_RegisterCallback() before calling HAL_SMARTCARD_DeInit() or HAL_SMARTCARD_Init() function.

When The compilation define USE_HAL_SMARTCARD_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

74.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx associated to the SmartCard.

- These parameters can be configured:
  - Baud Rate
  - Parity: parity should be enabled, frame Length is fixed to 8 bits plus parity
  - Receiver/transmitter modes
  - Synchronous mode (and if enabled, phase, polarity and last bit parameters)
  - Prescaler value
  - Guard bit time
  - NACK enabling or disabling on transmission error
- The following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - Time out enabling (and if activated, timeout value)
  - Block length
  - Auto-retry counter

The HAL_SMARTCARD_Init() API follows the USART synchronous configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:
- *HAL_SMARTCARD_Init()*
- *HAL_SMARTCARD_DeInit()*
- *HAL_SMARTCARD_MspInit()*
- *HAL_SMARTCARD_MspDeInit()*
- *HAL_SMARTCARD_RegisterCallback()*
- *HAL_SMARTCARD_UnRegisterCallback()*

74.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard. The USART should be configured as:
- 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
- 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.

1. There are two modes of transfer:
   a. Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   b. Non-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
   c. The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
2. Blocking mode APIs are :
   a. HAL_SMARTCARD_Transmit()
   b. HAL_SMARTCARD_Receive()
3. Non Blocking mode APIs with Interrupt are :
   a. HAL_SMARTCARD_Transmit_IT()
   b. HAL_SMARTCARD_Receive_IT()
   c. HAL_SMARTCARD_IRQHandler()
4. Non Blocking mode functions with DMA are :
   a. HAL_SMARTCARD_Transmit_DMA()
   b. HAL_SMARTCARD_Receive_DMA()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   a. HAL_SMARTCARD_TxCpltCallback()
   b. HAL_SMARTCARD_RxCpltCallback()
   c. HAL_SMARTCARD_ErrorCallback()

1. Non-Blocking mode transfers could be aborted using Abort API's :
   a. HAL_SMARTCARD_Abort()
   b. HAL_SMARTCARD_AbortTransmit()
   c. HAL_SMARTCARD_AbortReceive()
   d. HAL_SMARTCARD_Abort_IT()
   e. HAL_SMARTCARD_AbortTransmit_IT()
   f. HAL_SMARTCARD_AbortReceive_IT()
2. For Abort services based on interrupts (HAL_SMARTCARD_Abortxxx_IT), a set of Abort Complete Callbacks are provided:
   a. HAL_SMARTCARD_AbortCpltCallback()
   b. HAL_SMARTCARD_AbortTransmitCpltCallback()
   c. HAL_SMARTCARD_AbortReceiveCpltCallback()

3. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   a. Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed. Transfer is kept ongoing on SMARTCARD side. If user wants to abort it, Abort services should be called by user.
   b. Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Frame Error in Interrupt mode transmission, Overrun Error in Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_SMARTCARD_ErrorCallback() user callback is executed.

This section contains the following APIs:
- *HAL_SMARTCARD_Transmit()*
- *HAL_SMARTCARD_Receive()*
- *HAL_SMARTCARD_Transmit_IT()*
- *HAL_SMARTCARD_Receive_IT()*
- *HAL_SMARTCARD_Transmit_DMA()*
- *HAL_SMARTCARD_Receive_DMA()*
- *HAL_SMARTCARD_Abort()*
- *HAL_SMARTCARD_AbortTransmit()*
- *HAL_SMARTCARD_AbortReceive()*
- *HAL_SMARTCARD_Abort_IT()*
- *HAL_SMARTCARD_AbortTransmit_IT()*
- *HAL_SMARTCARD_AbortReceive_IT()*
- *HAL_SMARTCARD_IRQHandler()*
- *HAL_SMARTCARD_TxCpltCallback()*
- *HAL_SMARTCARD_RxCpltCallback()*
- *HAL_SMARTCARD_ErrorCallback()*
- *HAL_SMARTCARD_AbortCpltCallback()*
- *HAL_SMARTCARD_AbortTransmitCpltCallback()*
- *HAL_SMARTCARD_AbortReceiveCpltCallback()*

74.2.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard handle and also return Peripheral Errors occurred during communication process
- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() checks in run-time errors that could occur during communication.

This section contains the following APIs:
- *HAL_SMARTCARD_GetState()*
- *HAL_SMARTCARD_GetError()*

74.2.6 Detailed description of functions

HAL_SMARTCARD_Init

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD mode according to the specified parameters in the SMARTCARD_HandleTypeDef and initialize the associated handle.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_DeInit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD peripheral.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARD_MspInit

Function name

void HAL_SMARTCARD_MspInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Initialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_MspDeInit

Function name

void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

DeInitialize the SMARTCARD MSP.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_RegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID, pSMARTCARD_CallbackTypeDef pCallback)

Function description

Register a User SMARTCARD Callback To be used to override the weak predefined callback.

Parameters

- **hsmartcard:** smartcard handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SMARTCARD_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_SMARTCARD_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_SMARTCARD_ERROR_CB_ID Error Callback ID
  - HAL_SMARTCARD_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_SMARTCARD_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_SMARTCARD_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_SMARTCARD_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_SMARTCARD_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_SMARTCARD_MSPINIT_CB_ID MspInit Callback ID
  - HAL_SMARTCARD_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SMARTCARD_RegisterCallback() may be called before HAL_SMARTCARD_Init() in HAL_SMARTCARD_STATE_RESET to register callbacks for HAL_SMARTCARD_MSPINIT_CB_ID and HAL_SMARTCARD_MSPDEINIT_CB_ID

HAL_SMARTCARD_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMARTCARD_UnRegisterCallback (SMARTCARD_HandleTypeDef * hsmartcard, HAL_SMARTCARD_CallbackIDTypeDef CallbackID)

Function description

Unregister an SMARTCARD callback SMARTCARD callback is redirected to the weak predefined callback.

Parameters

- **hsmartcard:** smartcard handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SMARTCARD_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_SMARTCARD_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_SMARTCARD_ERROR_CB_ID Error Callback ID
  - HAL_SMARTCARD_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_SMARTCARD_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_SMARTCARD_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_SMARTCARD_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_SMARTCARD_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_SMARTCARD_MSPINIT_CB_ID MspInit Callback ID
  - HAL_SMARTCARD_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_SMARTCARD_UnRegisterCallback() may be called before HAL_SMARTCARD_Init() in HAL_SMARTCARD_STATE_RESET to un-register callbacks for HAL_SMARTCARD_MSPINIT_CB_ID and HAL_SMARTCARD_MSPDEINIT_CB_ID

HAL_SMARTCARD_Transmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_SMARTCARD_Receive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_SMARTCARD_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_TDR register is empty, i.e one interrupt per data to transmit.
- When FIFO mode is enabled, USART interrupt is generated whenever TXFIFO threshold reached. In that case the interrupt rate depends on TXFIFO threshold configuration.
- This function sets the hsmartcard->TxIsr function pointer according to the FIFO mode (data transmission processing depends on FIFO mode).

HAL_SMARTCARD_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- When FIFO mode is disabled, USART interrupt is generated whenever USART_RDR register can be read, i.e one interrupt per data to receive.
- When FIFO mode is enabled, USART interrupt is generated whenever RXFIFO threshold reached. In that case the interrupt rate depends on RXFIFO threshold configuration.
- This function sets the hsmartcard->RxIsr function pointer according to the FIFO mode (data reception processing depends on FIFO mode).

HAL_SMARTCARD_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsmartcard, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be sent.

Return values

- **HAL:** status

HAL_SMARTCARD_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **pData:** pointer to data buffer.
- **Size:** amount of data to be received.

Return values

- **HAL:** status

Notes

- The SMARTCARD-associated USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).

HAL_SMARTCARD_Abort

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortTransmit

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_AbortReceive

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SMARTCARD_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_Abort_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortTransmit_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_SMARTCARD_AbortReceive_IT (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable SMARTCARD Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SMARTCARD_IRQHandler

Function name

void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Handle SMARTCARD interrupt requests.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_TxCpltCallback

Function name

void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Tx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_RxCpltCallback

Function name

void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Rx Transfer completed callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_ErrorCallback

Function name

void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD error callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortCpltCallback

Function name

void HAL_SMARTCARD_AbortCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortTransmitCpltCallback

Function name

void HAL_SMARTCARD_AbortTransmitCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_AbortReceiveCpltCallback

Function name

void HAL_SMARTCARD_AbortReceiveCpltCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD Abort Receive Complete callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARD_GetState

Function name

HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle state.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle state

HAL_SMARTCARD_GetError

Function name

uint32_t HAL_SMARTCARD_GetError (const SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Return the SMARTCARD handle error code.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **SMARTCARD:** handle Error Code

74.3 SMARTCARD Firmware driver defines

The following section lists the various define and macros of the module.

74.3.1 SMARTCARD

SMARTCARD

SMARTCARD Clock Prescaler

SMARTCARD_PRESCALER_DIV1

fclk_pres = fclk

SMARTCARD_PRESCALER_DIV2

fclk_pres = fclk/2

SMARTCARD_PRESCALER_DIV4

fclk_pres = fclk/4

SMARTCARD_PRESCALER_DIV6

fclk_pres = fclk/6

SMARTCARD_PRESCALER_DIV8

fclk_pres = fclk/8

SMARTCARD_PRESCALER_DIV10

fclk_pres = fclk/10

SMARTCARD_PRESCALER_DIV12

fclk_pres = fclk/12

SMARTCARD_PRESCALER_DIV16

fclk_pres = fclk/16

SMARTCARD_PRESCALER_DIV32

fclk_pres = fclk/32

SMARTCARD_PRESCALER_DIV64

fclk_pres = fclk/64

SMARTCARD_PRESCALER_DIV128

fclk_pres = fclk/128

SMARTCARD_PRESCALER_DIV256

fclk_pres = fclk/256

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD frame phase on first clock transition

SMARTCARD_PHASE_2EDGE

SMARTCARD frame phase on second clock transition

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD frame low polarity

SMARTCARD_POLARITY_HIGH

SMARTCARD frame high polarity

SMARTCARD advanced feature Binary Data inversion

SMARTCARD_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

SMARTCARD_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

SMARTCARD advanced feature DMA Disable on Rx Error

SMARTCARD_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

SMARTCARD_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

SMARTCARD Error Code Definition

HAL_SMARTCARD_ERROR_NONE

No error

HAL_SMARTCARD_ERROR_PE

Parity error

HAL_SMARTCARD_ERROR_NE

Noise error

HAL_SMARTCARD_ERROR_FE

frame error

HAL_SMARTCARD_ERROR_ORE

Overrun error

HAL_SMARTCARD_ERROR_DMA

DMA transfer error

HAL_SMARTCARD_ERROR_RTO

Receiver TimeOut error

HAL_SMARTCARD_ERROR_INVALID_CALLBACK

Invalid Callback error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:
- Reset SMARTCARD handle states.

Parameters:
- __HANDLE__: SMARTCARD handle.

Return value:
- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:
- Flush the Smartcard Data registers.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.

Return value:
- None

__HAL_SMARTCARD_CLEAR_FLAG

Description:
- Clear the specified SMARTCARD pending flag.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  – SMARTCARD_CLEAR_PEF Parity error clear flag
  – SMARTCARD_CLEAR_FEF Framing error clear flag
  – SMARTCARD_CLEAR_NEF Noise detected clear flag
  – SMARTCARD_CLEAR_OREF OverRun error clear flag
  – SMARTCARD_CLEAR_IDLEF Idle line detected clear flag
  – SMARTCARD_CLEAR_TCF Transmission complete clear flag
  – SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag
  – SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  – SMARTCARD_CLEAR_EOBF End of block clear flag
  – SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear flag

Return value:
- None

__HAL_SMARTCARD_CLEAR_PEFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_NEFLAG

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_OREFLAG

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_CLEAR_IDLEFLAG

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

Return value:

- None

__HAL_SMARTCARD_GET_FLAG

Description:

- Check whether the specified Smartcard flag is set or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – SMARTCARD_FLAG_TCBGT Transmission complete before guard time flag (when flag available)
  – SMARTCARD_FLAG_REACK Receive enable acknowledge flag
  – SMARTCARD_FLAG_TEACK Transmit enable acknowledge flag
  – SMARTCARD_FLAG_BUSY Busy flag
  – SMARTCARD_FLAG_EOBF End of block flag
  – SMARTCARD_FLAG_RTOF Receiver timeout flag
  – SMARTCARD_FLAG_TXE Transmit data register empty flag
  – SMARTCARD_FLAG_TC Transmission complete flag
  – SMARTCARD_FLAG_RXNE Receive data register not empty flag
  – SMARTCARD_FLAG_IDLE Idle line detection flag
  – SMARTCARD_FLAG_ORE Overrun error flag
  – SMARTCARD_FLAG_NE Noise error flag
  – SMARTCARD_FLAG_FE Framing error flag
  – SMARTCARD_FLAG_PE Parity error flag
  – SMARTCARD_FLAG_TXFNF TXFIFO not full flag
  – SMARTCARD_FLAG_RXFNE RXFIFO not empty flag
  – SMARTCARD_FLAG_TXFE TXFIFO Empty flag
  – SMARTCARD_FLAG_RXFF RXFIFO Full flag
  – SMARTCARD_FLAG_RXFT SMARTCARD RXFIFO threshold flag
  – SMARTCARD_FLAG_TXFT SMARTCARD TXFIFO threshold flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SMARTCARD_ENABLE_IT

Description:

- Enable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_DISABLE_IT

Description:

- Disable the specified SmartCard interrupt.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
  – SMARTCARD_IT_EOB End of block interrupt
  – SMARTCARD_IT_RTO Receive timeout interrupt
  – SMARTCARD_IT_TXE Transmit data register empty interrupt
  – SMARTCARD_IT_TC Transmission complete interrupt
  – SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  – SMARTCARD_IT_RXNE Receive data register not empty interrupt
  – SMARTCARD_IT_IDLE Idle line detection interrupt
  – SMARTCARD_IT_PE Parity error interrupt
  – SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  – SMARTCARD_IT_TXFNF TX FIFO not full interruption
  – SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  – SMARTCARD_IT_RXFF RXFIFO full interruption
  – SMARTCARD_IT_TXFE TXFIFO empty interruption
  – SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  – SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- None

__HAL_SMARTCARD_GET_IT

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __INTERRUPT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
  - SMARTCARD_IT_EOB End of block interrupt
  - SMARTCARD_IT_RTO Receive timeout interrupt
  - SMARTCARD_IT_TXE Transmit data register empty interrupt
  - SMARTCARD_IT_TC Transmission complete interrupt
  - SMARTCARD_IT_TCBGT Transmission complete before guard time interrupt (when interruption available)
  - SMARTCARD_IT_RXNE Receive data register not empty interrupt
  - SMARTCARD_IT_IDLE Idle line detection interrupt
  - SMARTCARD_IT_PE Parity error interrupt
  - SMARTCARD_IT_ERR Error interrupt(frame error, noise error, overrun error)
  - SMARTCARD_IT_TXFNF TX FIFO not full interruption
  - SMARTCARD_IT_RXFNE RXFIFO not empty interruption
  - SMARTCARD_IT_RXFF RXFIFO full interruption
  - SMARTCARD_IT_TXFE TXFIFO empty interruption
  - SMARTCARD_IT_RXFT RXFIFO threshold reached interruption
  - SMARTCARD_IT_TXFT TXFIFO threshold reached interruption

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_SMARTCARD_CLEAR_IT

Description:

- Clear the specified SMARTCARD ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - SMARTCARD_CLEAR_PEF Parity error clear flag
  - SMARTCARD_CLEAR_FEF Framing error clear flag
  - SMARTCARD_CLEAR_NEF Noise detected clear flag
  - SMARTCARD_CLEAR_OREF OverRun error clear flag
  - SMARTCARD_CLEAR_IDLEF Idle line detection clear flag
  - SMARTCARD_CLEAR_TXFECF TXFIFO empty Clear Flag
  - SMARTCARD_CLEAR_TCF Transmission complete clear flag
  - SMARTCARD_CLEAR_TCBGTF Transmission complete before guard time clear flag (when flag available)
  - SMARTCARD_CLEAR_RTOF Receiver timeout clear flag
  - SMARTCARD_CLEAR_EOBF End of block clear flag

Return value:

- None

__HAL_SMARTCARD_SEND_REQ

Description:
- Set a specific SMARTCARD request flag.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  - SMARTCARD_RXDATA_FLUSH_REQUEST Receive data flush Request
  - SMARTCARD_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:
- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:
- Enable the SMARTCARD one bit sample method.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.

Return value:
- None

__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE

Description:
- Disable the SMARTCARD one bit sample method.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.

Return value:
- None

__HAL_SMARTCARD_ENABLE

Description:
- Enable the USART associated to the SMARTCARD Handle.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.

Return value:
- None

__HAL_SMARTCARD_DISABLE

Description:
- Disable the USART associated to the SMARTCARD Handle.

Parameters:
- __HANDLE__: specifies the SMARTCARD Handle.

Return value:
- None

SMARTCARD interruptions flags mask

SMARTCARD_IT_MASK

SMARTCARD interruptions flags mask

SMARTCARD_CR_MASK

SMARTCARD control register mask

SMARTCARD_CR_POS

SMARTCARD control register position

SMARTCARD_ISR_MASK

SMARTCARD ISR register mask

SMARTCARD_ISR_POS

SMARTCARD ISR register position

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD frame last data bit clock pulse not output to SCLK pin

SMARTCARD_LASTBIT_ENABLE

SMARTCARD frame last data bit clock pulse output to SCLK pin

SMARTCARD Transfer Mode

SMARTCARD_MODE_RX

SMARTCARD RX mode

SMARTCARD_MODE_TX

SMARTCARD TX mode

SMARTCARD_MODE_TX_RX

SMARTCARD RX and TX mode

SMARTCARD advanced feature MSB first

SMARTCARD_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

SMARTCARD_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

SMARTCARD NACK Enable

SMARTCARD_NACK_DISABLE

SMARTCARD NACK transmission disabled

SMARTCARD_NACK_ENABLE

SMARTCARD NACK transmission enabled

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE

SMARTCARD frame one-bit sample disabled

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD frame one-bit sample enabled

SMARTCARD advanced feature Overrun Disable

SMARTCARD_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

SMARTCARD_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD frame even parity

SMARTCARD_PARITY_ODD

SMARTCARD frame odd parity

SMARTCARD Request Parameters

SMARTCARD_RXDATA_FLUSH_REQUEST

Receive data flush request

SMARTCARD_TXDATA_FLUSH_REQUEST

Transmit data flush request

SMARTCARD advanced feature RX pin active level inversion

SMARTCARD_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

SMARTCARD_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

SMARTCARD advanced feature RX TX pins swap

SMARTCARD_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

SMARTCARD_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

SMARTCARD State Code Definition

HAL_SMARTCARD_STATE_RESET

Peripheral is not initialized. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_READY

Peripheral Initialized and ready for use. Value is allowed for gState and RxState

HAL_SMARTCARD_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_SMARTCARD_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_SMARTCARD_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState. Value is result of combination (Or) between gState and RxState values

HAL_SMARTCARD_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_SMARTCARD_STATE_ERROR

Error Value is allowed for gState only

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD frame with 0.5 stop bit

SMARTCARD_STOPBITS_1_5

SMARTCARD frame with 1.5 stop bits

SMARTCARD Timeout Enable

SMARTCARD_TIMEOUT_DISABLE

SMARTCARD receiver timeout disabled

SMARTCARD_TIMEOUT_ENABLE

SMARTCARD receiver timeout enabled

SMARTCARD advanced feature TX pin active level inversion

SMARTCARD_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

SMARTCARD_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

SMARTCARD frame length

75 HAL SMARTCARD Extension Driver

75.1 SMARTCARDEx Firmware driver API description

The following section lists the various functions of the SMARTCARDEx library.

75.1.1 SMARTCARD peripheral extended features

The Extended SMARTCARD HAL driver can be used as follows:

1. After having configured the SMARTCARD basic features with HAL_SMARTCARD_Init(), then program SMARTCARD advanced features if required (TX/RX pins swap, TimeOut, auto-retry counter,...) in the hsmartcard AdvancedInit structure.
2. FIFO mode enabling/disabling and RX/TX FIFO threshold programming.

Note: *When SMARTCARD operates in FIFO mode, FIFO mode must be enabled prior starting RX/TX transfers. Also RX/TX FIFO thresholds must be configured prior starting RX/TX transfers.*

75.1.2 Peripheral Control functions

This subsection provides a set of functions allowing to initialize the SMARTCARD.

- HAL_SMARTCARDEx_BlockLength_Config() API allows to configure the Block Length on the fly
- HAL_SMARTCARDEx_TimeOut_Config() API allows to configure the receiver timeout value on the fly
- HAL_SMARTCARDEx_EnableReceiverTimeOut() API enables the receiver timeout feature
- HAL_SMARTCARDEx_DisableReceiverTimeOut() API disables the receiver timeout feature

This section contains the following APIs:

- ***HAL_SMARTCARDEx_BlockLength_Config()***
- ***HAL_SMARTCARDEx_TimeOut_Config()***
- ***HAL_SMARTCARDEx_EnableReceiverTimeOut()***
- ***HAL_SMARTCARDEx_DisableReceiverTimeOut()***

75.1.3 IO operation functions

This subsection provides a set of FIFO mode related callback functions.

1. TX/RX Fifos Callbacks:
   – HAL_SMARTCARDEx_RxFifoFullCallback()
   – HAL_SMARTCARDEx_TxFifoEmptyCallback()

This section contains the following APIs:

- ***HAL_SMARTCARDEx_RxFifoFullCallback()***
- ***HAL_SMARTCARDEx_TxFifoEmptyCallback()***

75.1.4 Peripheral FIFO Control functions

This subsection provides a set of functions allowing to control the SMARTCARD FIFO feature.

- HAL_SMARTCARDEx_EnableFifoMode() API enables the FIFO mode
- HAL_SMARTCARDEx_DisableFifoMode() API disables the FIFO mode
- HAL_SMARTCARDEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_SMARTCARDEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_SMARTCARDEx_EnableFifoMode()***
- ***HAL_SMARTCARDEx_DisableFifoMode()***
- ***HAL_SMARTCARDEx_SetTxFifoThreshold()***
- ***HAL_SMARTCARDEx_SetRxFifoThreshold()***

75.1.5 Detailed description of functions

HAL_SMARTCARDEx_BlockLength_Config

Function name

void HAL_SMARTCARDEx_BlockLength_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint8_t BlockLength)

Function description

Update on the fly the SMARTCARD block length in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **BlockLength:** SMARTCARD block length (8-bit long at most)

Return values

- **None:**

HAL_SMARTCARDEx_TimeOut_Config

Function name

void HAL_SMARTCARDEx_TimeOut_Config (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t TimeOutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
- **TimeOutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.

Return values

- **None:**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_RxFifoFullCallback

Function name

void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD RX Fifo full callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name

void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD TX Fifo empty callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  – SMARTCARD_TXFIFO_THRESHOLD_1_8
  – SMARTCARD_TXFIFO_THRESHOLD_1_4
  – SMARTCARD_TXFIFO_THRESHOLD_1_2
  – SMARTCARD_TXFIFO_THRESHOLD_3_4
  – SMARTCARD_TXFIFO_THRESHOLD_7_8
  – SMARTCARD_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  – SMARTCARD_RXFIFO_THRESHOLD_1_8
  – SMARTCARD_RXFIFO_THRESHOLD_1_4
  – SMARTCARD_RXFIFO_THRESHOLD_1_2
  – SMARTCARD_RXFIFO_THRESHOLD_3_4
  – SMARTCARD_RXFIFO_THRESHOLD_7_8
  – SMARTCARD_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

75.2 SMARTCARDEx Firmware driver defines

The following section lists the various define and macros of the module.

75.2.1 SMARTCARDEx

SMARTCARDEx

SMARTCARD advanced feature initialization type

SMARTCARD_ADVFEATURE_NO_INIT

No advanced feature initialization

SMARTCARD_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

SMARTCARD_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

SMARTCARD_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

SMARTCARD_ADVFEATURE_SWAP_INIT

TX/RX pins swap

SMARTCARD_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

SMARTCARD_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

SMARTCARD_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

SMARTCARD_ADVFEATURE_TXCOMPLETION

TX completion indication before of after guard time

SMARTCARD FIFO mode

SMARTCARD_FIFOMODE_DISABLE

FIFO mode disable

SMARTCARD_FIFOMODE_ENABLE

FIFO mode enable

SMARTCARD Flags

SMARTCARD_FLAG_TCBGT

SMARTCARD transmission complete before guard time completion

SMARTCARD_FLAG_REACK

SMARTCARD receive enable acknowledge flag

SMARTCARD_FLAG_TEACK

SMARTCARD transmit enable acknowledge flag

SMARTCARD_FLAG_BUSY
SMARTCARD busy flag

SMARTCARD_FLAG_EOBF
SMARTCARD end of block flag

SMARTCARD_FLAG_RTOF
SMARTCARD receiver timeout flag

SMARTCARD_FLAG_TXE
SMARTCARD transmit data register empty

SMARTCARD_FLAG_TXFNF
SMARTCARD TXFIFO not full

SMARTCARD_FLAG_TC
SMARTCARD transmission complete

SMARTCARD_FLAG_RXNE
SMARTCARD read data register not empty

SMARTCARD_FLAG_RXFNE
SMARTCARD RXFIFO not empty

SMARTCARD_FLAG_IDLE
SMARTCARD idle line detection

SMARTCARD_FLAG_ORE
SMARTCARD overrun error

SMARTCARD_FLAG_NE
SMARTCARD noise error

SMARTCARD_FLAG_FE
SMARTCARD frame error

SMARTCARD_FLAG_PE
SMARTCARD parity error

SMARTCARD_FLAG_TXFE
SMARTCARD TXFIFO Empty flag

SMARTCARD_FLAG_RXFF
SMARTCARD RXFIFO Full flag

SMARTCARD_FLAG_RXFT
SMARTCARD RXFIFO threshold flag

SMARTCARD_FLAG_TXFT
SMARTCARD TXFIFO threshold flag

SMARTCARD Interrupts Definition

SMARTCARD_IT_PE
SMARTCARD parity error interruption

SMARTCARD_IT_TXE
SMARTCARD transmit data register empty interruption

SMARTCARD_IT_TXFNF
SMARTCARD TX FIFO not full interruption

SMARTCARD_IT_TC
SMARTCARD transmission complete interruption

SMARTCARD_IT_RXNE
SMARTCARD read data register not empty interruption

SMARTCARD_IT_RXFNE
SMARTCARD RXFIFO not empty interruption

SMARTCARD_IT_IDLE
SMARTCARD idle line detection interruption

SMARTCARD_IT_ERR
SMARTCARD error interruption

SMARTCARD_IT_ORE
SMARTCARD overrun error interruption

SMARTCARD_IT_NE
SMARTCARD noise error interruption

SMARTCARD_IT_FE
SMARTCARD frame error interruption

SMARTCARD_IT_EOB
SMARTCARD end of block interruption

SMARTCARD_IT_RTO
SMARTCARD receiver timeout interruption

SMARTCARD_IT_TCBGT
SMARTCARD transmission complete before guard time completion interruption

SMARTCARD_IT_RXFF
SMARTCARD RXFIFO full interruption

SMARTCARD_IT_TXFE
SMARTCARD TXFIFO empty interruption

SMARTCARD_IT_RXFT
SMARTCARD RXFIFO threshold reached interruption

SMARTCARD_IT_TXFT
SMARTCARD TXFIFO threshold reached interruption

SMARTCARD Interruption Clear Flags

SMARTCARD_CLEAR_PEF
SMARTCARD parity error clear flag

SMARTCARD_CLEAR_FEF
SMARTCARD framing error clear flag

SMARTCARD_CLEAR_NEF
SMARTCARD noise error detected clear flag

SMARTCARD_CLEAR_OREF

SMARTCARD overrun error clear flag

SMARTCARD_CLEAR_IDLEF

SMARTCARD idle line detected clear flag

SMARTCARD_CLEAR_TXFECF

TXFIFO empty Clear Flag

SMARTCARD_CLEAR_TCF

SMARTCARD transmission complete clear flag

SMARTCARD_CLEAR_TCBGTF

SMARTCARD transmission complete before guard time completion clear flag

SMARTCARD_CLEAR_RTOF

SMARTCARD receiver time out clear flag

SMARTCARD_CLEAR_EOBF

SMARTCARD end of block clear flag

SMARTCARD RXFIFO threshold level

SMARTCARD_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

SMARTCARD_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

SMARTCARD_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

SMARTCARD_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

SMARTCARD Transmission Completion Indication

SMARTCARD_TCBGT

SMARTCARD transmission complete before guard time

SMARTCARD_TC

SMARTCARD transmission complete (flag raised when guard time has elapsed)

SMARTCARD TXFIFO threshold level

SMARTCARD_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

SMARTCARD_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

SMARTCARD_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

SMARTCARD_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

76 HAL SMBUS Generic Driver

76.1 SMBUS Firmware driver registers structures

76.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32u5xx_hal_smbus.h

Data Fields

- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t AddressingMode***
- ***uint32_t DualAddressMode***
- ***uint32_t OwnAddress2***
- ***uint32_t OwnAddress2Masks***
- ***uint32_t GeneralCallMode***
- ***uint32_t NoStretchMode***
- ***uint32_t PacketErrorCheckMode***
- ***uint32_t PeripheralMode***
- ***uint32_t SMBusTimeout***

Field Documentation

- ***uint32_t SMBUS_InitTypeDef::Timing***
  Specifies the SMBUS_TIMINGR_register value. This parameter calculated by referring to SMBUS initialization section in Reference manual
- ***uint32_t SMBUS_InitTypeDef::AnalogFilter***
  Specifies if Analog Filter is enable or not. This parameter can be a value of ***SMBUS_Analog_Filter***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress1***
  Specifies the first device own address. This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::AddressingMode***
  Specifies addressing mode selected. This parameter can be a value of ***SMBUS_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::DualAddressMode***
  Specifies if dual addressing mode is selected. This parameter can be a value of ***SMBUS_dual_addressing_mode***
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2***
  Specifies the second device own address if dual addressing mode is selected This parameter can be a 7-bit address.
- ***uint32_t SMBUS_InitTypeDef::OwnAddress2Masks***
  Specifies the acknowledge mask address second device own address if dual addressing mode is selected This parameter can be a value of ***SMBUS_own_address2_masks***.
- ***uint32_t SMBUS_InitTypeDef::GeneralCallMode***
  Specifies if general call mode is selected. This parameter can be a value of ***SMBUS_general_call_addressing_mode***.
- ***uint32_t SMBUS_InitTypeDef::NoStretchMode***
  Specifies if nostretch mode is selected. This parameter can be a value of ***SMBUS_nostretch_mode***
- ***uint32_t SMBUS_InitTypeDef::PacketErrorCheckMode***
  Specifies if Packet Error Check mode is selected. This parameter can be a value of ***SMBUS_packet_error_check_mode***
- ***uint32_t SMBUS_InitTypeDef::PeripheralMode***
  Specifies which mode of Periphal is selected. This parameter can be a value of ***SMBUS_peripheral_mode***
- ***uint32_t SMBUS_InitTypeDef::SMBusTimeout***
  Specifies the content of the 32 Bits SMBUS_TIMEOUT_register value. (Enable bits and different timeout values) This parameter calculated by referring to SMBUS initialization section in Reference manual

76.1.2 __SMBUS_HandleTypeDef

__SMBUS_HandleTypeDef is defined in the stm32u5xx_hal_smbus.h

Data Fields

- ***I2C_TypeDef * Instance***
- ***SMBUS_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***uint16_t XferSize***
- ***__IO uint16_t XferCount***
- ***__IO uint32_t XferOptions***
- ***__IO uint32_t PreviousState***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***void(* MasterTxCpltCallback***
- ***void(* MasterRxCpltCallback***
- ***void(* SlaveTxCpltCallback***
- ***void(* SlaveRxCpltCallback***
- ***void(* ListenCpltCallback***
- ***void(* ErrorCallback***
- ***void(* AddrCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***I2C_TypeDef* __SMBUS_HandleTypeDef::Instance***
  SMBUS registers base address
- ***SMBUS_InitTypeDef __SMBUS_HandleTypeDef::Init***
  SMBUS communication parameters
- ***uint8_t* __SMBUS_HandleTypeDef::pBuffPtr***
  Pointer to SMBUS transfer buffer
- ***uint16_t __SMBUS_HandleTypeDef::XferSize***
  SMBUS transfer size
- ***__IO uint16_t __SMBUS_HandleTypeDef::XferCount***
  SMBUS transfer counter
- ***__IO uint32_t __SMBUS_HandleTypeDef::XferOptions***
  SMBUS transfer options
- ***__IO uint32_t __SMBUS_HandleTypeDef::PreviousState***
  SMBUS communication Previous state
- ***HAL_LockTypeDef __SMBUS_HandleTypeDef::Lock***
  SMBUS locking object
- ***__IO uint32_t __SMBUS_HandleTypeDef::State***
  SMBUS communication state
- ***__IO uint32_t __SMBUS_HandleTypeDef::ErrorCode***
  SMBUS Error code
- ***void(* __SMBUS_HandleTypeDef::MasterTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::MasterRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Master Rx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveTxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Tx Transfer completed callback
- ***void(* __SMBUS_HandleTypeDef::SlaveRxCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Slave Rx Transfer completed callback

- ***void(* __SMBUS_HandleTypeDef::ListenCpltCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Listen Complete callback
- ***void(* __SMBUS_HandleTypeDef::ErrorCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Error callback
- ***void(* __SMBUS_HandleTypeDef::AddrCallback)(struct __SMBUS_HandleTypeDef *hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)***
  SMBUS Slave Address Match callback
- ***void(* __SMBUS_HandleTypeDef::MspInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp Init callback
- ***void(* __SMBUS_HandleTypeDef::MspDeInitCallback)(struct __SMBUS_HandleTypeDef *hsmbus)***
  SMBUS Msp DeInit callback

76.2 SMBUS Firmware driver API description

The following section lists the various functions of the SMBUS library.

76.2.1 How to use this driver

The SMBUS HAL driver can be used as follows:

1. Declare a SMBUS_HandleTypeDef handle structure, for example: SMBUS_HandleTypeDef hsmbus;
2. Initialize the SMBUS low level resources by implementing the HAL_SMBUS_MspInit() API:
   a. Enable the SMBUSx interface clock
   b. SMBUS pins configuration
      - Enable the clock for the SMBUS GPIOs
      - Configure SMBUS pins as alternate function open-drain
   c. NVIC configuration if you need to use interrupt process
      - Configure the SMBUSx interrupt priority
      - Enable the NVIC SMBUS IRQ Channel
3. Configure the Communication Clock Timing, Bus Timeout, Own Address1, Master Addressing mode, Dual Addressing mode, Own Address2, Own Address2 Mask, General call, Nostretch mode, Peripheral mode and Packet Error Check mode in the hsmbus Init structure.
4. Initialize the SMBUS registers by calling the HAL_SMBUS_Init() API:
   – These API's configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMBUS_MspInit(&hsmbus) API.
5. To check if target device is ready for communication, use the function HAL_SMBUS_IsDeviceReady()
6. For SMBUS IO operations, only one mode of operations is available within this driver

Interrupt mode IO operation

- Transmit in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_MasterTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterTxCpltCallback()
- Receive in master/host SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Master_Receive_IT()
  – At reception end of transfer HAL_SMBUS_MasterRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_MasterRxCpltCallback()
- Abort a master/host SMBUS process communication with Interrupt using HAL_SMBUS_Master_Abort_IT()
  – The associated previous transfer callback is called at the end of abort process
  – mean HAL_SMBUS_MasterTxCpltCallback() in case of previous state was master transmit
  – mean HAL_SMBUS_MasterRxCpltCallback() in case of previous state was master receive

- Enable/disable the Address listen mode in slave/device or host/slave SMBUS mode using HAL_SMBUS_EnableListen_IT() HAL_SMBUS_DisableListen_IT()
  – When address slave/device SMBUS match, HAL_SMBUS_AddrCallback() is executed and users can add their own code to check the Address Match Code and the transmission direction request by master/host (Write/Read).
  – At Listen mode end HAL_SMBUS_ListenCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ListenCpltCallback()
- Transmit in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Transmit_IT()
  – At transmission end of transfer HAL_SMBUS_SlaveTxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveTxCpltCallback()
- Receive in slave/device SMBUS mode an amount of data in non-blocking mode using HAL_SMBUS_Slave_Receive_IT()
  – At reception end of transfer HAL_SMBUS_SlaveRxCpltCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_SlaveRxCpltCallback()
- Enable/Disable the SMBUS alert mode using HAL_SMBUS_EnableAlert_IT() or HAL_SMBUS_DisableAlert_IT()
  – When SMBUS Alert is generated HAL_SMBUS_ErrorCallback() is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Alert Error Code using function HAL_SMBUS_GetError()
- Get HAL state machine or error values using HAL_SMBUS_GetState() or HAL_SMBUS_GetError()
- In case of transfer Error, HAL_SMBUS_ErrorCallback() function is executed and users can add their own code by customization of function pointer HAL_SMBUS_ErrorCallback() to check the Error Code using function HAL_SMBUS_GetError()

SMBUS HAL driver macros list

Below the list of most used macros in SMBUS HAL driver.

- __HAL_SMBUS_ENABLE: Enable the SMBUS peripheral
- __HAL_SMBUS_DISABLE: Disable the SMBUS peripheral
- __HAL_SMBUS_GET_FLAG: Check whether the specified SMBUS flag is set or not
- __HAL_SMBUS_CLEAR_FLAG: Clear the specified SMBUS pending flag
- __HAL_SMBUS_ENABLE_IT: Enable the specified SMBUS interrupt
- __HAL_SMBUS_DISABLE_IT: Disable the specified SMBUS interrupt

Callback registration

The compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SMBUS_RegisterCallback() or HAL_SMBUS_RegisterAddrCallback() to register an interrupt callback.

Function HAL_SMBUS_RegisterCallback() allows to register following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.
- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

For specific callback AddrCallback use dedicated register callbacks : HAL_SMBUS_RegisterAddrCallback.

Use function HAL_SMBUS_UnRegisterCallback to reset a callback to the default weak function. HAL_SMBUS_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- MasterTxCpltCallback : callback for Master transmission end of transfer.

- MasterRxCpltCallback : callback for Master reception end of transfer.
- SlaveTxCpltCallback : callback for Slave transmission end of transfer.
- SlaveRxCpltCallback : callback for Slave reception end of transfer.
- ListenCpltCallback : callback for end of listen mode.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

For callback AddrCallback use dedicated register callbacks : HAL_SMBUS_UnRegisterAddrCallback.

By default, after the HAL_SMBUS_Init() and when the state is HAL_I2C_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SMBUS_MasterTxCpltCallback(), HAL_SMBUS_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SMBUS_Init()/ HAL_SMBUS_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2C_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2C_STATE_READY or HAL_I2C_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SMBUS_RegisterCallback() before calling HAL_SMBUS_DeInit() or HAL_SMBUS_Init() function.

When the compilation flag USE_HAL_SMBUS_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

Note: You can refer to the SMBUS HAL driver header file for more useful macros

76.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the SMBUSx peripheral:

- User must Implement HAL_SMBUS_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, IT and NVIC).
- Call the function HAL_SMBUS_Init() to configure the selected device with the selected configuration:
  - Clock Timing
  - Bus Timeout
  - Analog Filer mode
  - Own Address 1
  - Addressing mode (Master, Slave)
  - Dual Addressing mode
  - Own Address 2
  - Own Address 2 Mask
  - General call mode
  - Nostretch mode
  - Packet Error Check mode
  - Peripheral mode
- Call the function HAL_SMBUS_DeInit() to restore the default configuration of the selected SMBUSx peripheral.
- Enable/Disable Analog/Digital filters with HAL_SMBUS_ConfigAnalogFilter() and HAL_SMBUS_ConfigDigitalFilter().

This section contains the following APIs:

- ***HAL_SMBUS_Init()***
- ***HAL_SMBUS_DeInit()***
- ***HAL_SMBUS_MspInit()***
- ***HAL_SMBUS_MspDeInit()***
- ***HAL_SMBUS_ConfigAnalogFilter()***
- ***HAL_SMBUS_ConfigDigitalFilter()***
- ***HAL_SMBUS_RegisterCallback()***

- *HAL_SMBUS_UnRegisterCallback()*
- *HAL_SMBUS_RegisterAddrCallback()*
- *HAL_SMBUS_UnRegisterAddrCallback()*

76.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMBUS data transfers.

1. Blocking mode function to check if device is ready for usage is :
   – HAL_SMBUS_IsDeviceReady()
2. There is only one mode of transfer:
   – Non-Blocking mode : The communication is performed using Interrupts. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated SMBUS IRQ when using Interrupt mode.
3. Non-Blocking mode functions with Interrupt are :
   – HAL_SMBUS_Master_Transmit_IT()
   – HAL_SMBUS_Master_Receive_IT()
   – HAL_SMBUS_Slave_Transmit_IT()
   – HAL_SMBUS_Slave_Receive_IT()
   – HAL_SMBUS_EnableListen_IT() or alias HAL_SMBUS_EnableListen_IT()
   – HAL_SMBUS_DisableListen_IT()
   – HAL_SMBUS_EnableAlert_IT()
   – HAL_SMBUS_DisableAlert_IT()
4. A set of Transfer Complete Callbacks are provided in non-Blocking mode:
   – HAL_SMBUS_MasterTxCpltCallback()
   – HAL_SMBUS_MasterRxCpltCallback()
   – HAL_SMBUS_SlaveTxCpltCallback()
   – HAL_SMBUS_SlaveRxCpltCallback()
   – HAL_SMBUS_AddrCallback()
   – HAL_SMBUS_ListenCpltCallback()
   – HAL_SMBUS_ErrorCallback()

This section contains the following APIs:

- *HAL_SMBUS_Master_Transmit_IT()*
- *HAL_SMBUS_Master_Receive_IT()*
- *HAL_SMBUS_Master_Abort_IT()*
- *HAL_SMBUS_Slave_Transmit_IT()*
- *HAL_SMBUS_Slave_Receive_IT()*
- *HAL_SMBUS_EnableListen_IT()*
- *HAL_SMBUS_DisableListen_IT()*
- *HAL_SMBUS_EnableAlert_IT()*
- *HAL_SMBUS_DisableAlert_IT()*
- *HAL_SMBUS_IsDeviceReady()*

76.2.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- *HAL_SMBUS_GetState()*
- *HAL_SMBUS_GetError()*

76.2.5 Detailed description of functions

HAL_SMBUS_Init

Function name

HAL_StatusTypeDef HAL_SMBUS_Init (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS according to the specified parameters in the SMBUS_InitTypeDef and initialize the associated handle.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DeInit

Function name

HAL_StatusTypeDef HAL_SMBUS_DeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS peripheral.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_MspInit

Function name

void HAL_SMBUS_MspInit (SMBUS_HandleTypeDef * hsmbus)

Function description

Initialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MspDeInit

Function name

void HAL_SMBUS_MspDeInit (SMBUS_HandleTypeDef * hsmbus)

Function description

DeInitialize the SMBUS MSP.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ConfigAnalogFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigAnalogFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t AnalogFilter)

Function description

Configure Analog noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **AnalogFilter:** This parameter can be one of the following values:
  – SMBUS_ANALOGFILTER_ENABLE
  – SMBUS_ANALOGFILTER_DISABLE

Return values

- **HAL:** status

HAL_SMBUS_ConfigDigitalFilter

Function name

HAL_StatusTypeDef HAL_SMBUS_ConfigDigitalFilter (SMBUS_HandleTypeDef * hsmbus, uint32_t DigitalFilter)

Function description

Configure Digital noise filter.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DigitalFilter:** Coefficient of digital noise filter between Min_Data=0x00 and Max_Data=0x0F.

Return values

- **HAL:** status

HAL_SMBUS_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID, pSMBUS_CallbackTypeDef pCallback)

Function description

Register a User SMBUS Callback To be used instead of the weak predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_SMBUS_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  – HAL_SMBUS_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  – HAL_SMBUS_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  – HAL_SMBUS_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  – HAL_SMBUS_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  – HAL_SMBUS_ERROR_CB_ID Error callback ID
  – HAL_SMBUS_MSPINIT_CB_ID MspInit callback ID
  – HAL_SMBUS_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SMBUS_RegisterCallback() may be called before HAL_SMBUS_Init() in HAL_SMBUS_STATE_RESET to register callbacks for HAL_SMBUS_MSPINIT_CB_ID and HAL_SMBUS_MSPDEINIT_CB_ID.

HAL_SMBUS_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterCallback (SMBUS_HandleTypeDef * hsmbus, HAL_SMBUS_CallbackIDTypeDef CallbackID)

Function description

Unregister an SMBUS Callback SMBUS callback is redirected to the weak predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  – HAL_SMBUS_MASTER_TX_COMPLETE_CB_ID Master Tx Transfer completed callback ID
  – HAL_SMBUS_MASTER_RX_COMPLETE_CB_ID Master Rx Transfer completed callback ID
  – HAL_SMBUS_SLAVE_TX_COMPLETE_CB_ID Slave Tx Transfer completed callback ID
  – HAL_SMBUS_SLAVE_RX_COMPLETE_CB_ID Slave Rx Transfer completed callback ID
  – HAL_SMBUS_LISTEN_COMPLETE_CB_ID Listen Complete callback ID
  – HAL_SMBUS_ERROR_CB_ID Error callback ID
  – HAL_SMBUS_MSPINIT_CB_ID MspInit callback ID
  – HAL_SMBUS_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

Notes

- The HAL_SMBUS_UnRegisterCallback() may be called before HAL_SMBUS_Init() in HAL_SMBUS_STATE_RESET to un-register callbacks for HAL_SMBUS_MSPINIT_CB_ID and HAL_SMBUS_MSPDEINIT_CB_ID

HAL_SMBUS_RegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_RegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus, pSMBUS_AddrCallbackTypeDef pCallback)

Function description

Register the Slave Address Match SMBUS Callback To be used instead of the weak HAL_SMBUS_AddrCallback() predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pCallback:** pointer to the Address Match Callback function

Return values

- **HAL:** status

HAL_SMBUS_UnRegisterAddrCallback

Function name

HAL_StatusTypeDef HAL_SMBUS_UnRegisterAddrCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

UnRegister the Slave Address Match SMBUS Callback Info Ready SMBUS Callback is redirected to the weak HAL_SMBUS_AddrCallback() predefined callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_IsDeviceReady

Function name

HAL_StatusTypeDef HAL_SMBUS_IsDeviceReady (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

Function description

Check if target device is ready for communication.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **Trials:** Number of trials
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SMBUS_Master_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in master/host SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Master_Abort_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Master_Abort_IT (SMBUS_HandleTypeDef * hsmbus, uint16_t DevAddress)

Function description

Abort a master/host SMBUS process communication with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **DevAddress:** Target device address: The device 7 bits address value in datasheet must be shifted to the left before calling the interface

Return values

- **HAL:** status

Notes

- This abort can be called only if state is ready

HAL_SMBUS_Slave_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Transmit_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Transmit in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_Slave_Receive_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_Slave_Receive_IT (SMBUS_HandleTypeDef * hsmbus, uint8_t * pData, uint16_t Size, uint32_t XferOptions)

Function description

Receive in slave/device SMBUS mode an amount of data in non-blocking mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **XferOptions:** Options of Transfer, value of SMBUS XferOptions definition

Return values

- **HAL:** status

HAL_SMBUS_EnableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_DisableAlert_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableAlert_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the SMBUS alert mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUS_EnableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_EnableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_DisableListen_IT

Function name

HAL_StatusTypeDef HAL_SMBUS_DisableListen_IT (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable the Address listen mode with Interrupt.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** status

HAL_SMBUS_EV_IRQHandler

Function name

void HAL_SMBUS_EV_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS event interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ER_IRQHandler

Function name

void HAL_SMBUS_ER_IRQHandler (SMBUS_HandleTypeDef * hsmbus)

Function description

Handle SMBUS error interrupt request.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterTxCpltCallback

Function name

void HAL_SMBUS_MasterTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_MasterRxCpltCallback

Function name

void HAL_SMBUS_MasterRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Master Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveTxCpltCallback

Function name

void HAL_SMBUS_SlaveTxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Tx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_SlaveRxCpltCallback

Function name

void HAL_SMBUS_SlaveRxCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Slave Rx Transfer completed callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_AddrCallback

Function name

void HAL_SMBUS_AddrCallback (SMBUS_HandleTypeDef * hsmbus, uint8_t TransferDirection, uint16_t AddrMatchCode)

Function description

Slave Address Match callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.
- **TransferDirection:** Master request Transfer Direction (Write/Read)
- **AddrMatchCode:** Address Match Code

Return values

- **None:**

HAL_SMBUS_ListenCpltCallback

Function name

void HAL_SMBUS_ListenCpltCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

Listen Complete callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_ErrorCallback

Function name

void HAL_SMBUS_ErrorCallback (SMBUS_HandleTypeDef * hsmbus)

Function description

SMBUS error callback.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **None:**

HAL_SMBUS_GetState

Function name

uint32_t HAL_SMBUS_GetState (const SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS handle state.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **HAL:** state

HAL_SMBUS_GetError

Function name

uint32_t HAL_SMBUS_GetError (const SMBUS_HandleTypeDef * hsmbus)

Function description

Return the SMBUS error code.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS.

Return values

- **SMBUS:** Error Code

76.3 SMBUS Firmware driver defines

The following section lists the various define and macros of the module.

76.3.1 SMBUS

SMBUS

SMBUS addressing mode

SMBUS_ADDRESSINGMODE_7BIT

SMBUS Analog Filter

SMBUS_ANALOGFILTER_ENABLE

SMBUS_ANALOGFILTER_DISABLE

SMBUS dual addressing mode

SMBUS_DUALADDRESS_DISABLE

SMBUS_DUALADDRESS_ENABLE

SMBUS Error Code definition

HAL_SMBUS_ERROR_NONE

No error

HAL_SMBUS_ERROR_BERR

BERR error

HAL_SMBUS_ERROR_ARLO

ARLO error

HAL_SMBUS_ERROR_ACKF

ACKF error

HAL_SMBUS_ERROR_OVR

OVR error

HAL_SMBUS_ERROR_HALTIMEOUT

Timeout error

HAL_SMBUS_ERROR_BUSTIMEOUT

Bus Timeout error

HAL_SMBUS_ERROR_ALERT

Alert error

HAL_SMBUS_ERROR_PECERR

PEC error

HAL_SMBUS_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_SMBUS_ERROR_INVALID_PARAM

Invalid Parameters error

SMBUS Exported Macros

__HAL_SMBUS_RESET_HANDLE_STATE

Description:

- Reset SMBUS handle state.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.

Return value:

- None

__HAL_SMBUS_ENABLE_IT

Description:

- Enable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  – SMBUS_IT_ERRI Errors interrupt enable
  – SMBUS_IT_TCI Transfer complete interrupt enable
  – SMBUS_IT_STOPI STOP detection interrupt enable
  – SMBUS_IT_NACKI NACK received interrupt enable
  – SMBUS_IT_ADDRI Address match interrupt enable
  – SMBUS_IT_RXI RX interrupt enable
  – SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_DISABLE_IT

Description:

- Disable the specified SMBUS interrupts.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  – SMBUS_IT_ERRI Errors interrupt enable
  – SMBUS_IT_TCI Transfer complete interrupt enable
  – SMBUS_IT_STOPI STOP detection interrupt enable
  – SMBUS_IT_NACKI NACK received interrupt enable
  – SMBUS_IT_ADDRI Address match interrupt enable
  – SMBUS_IT_RXI RX interrupt enable
  – SMBUS_IT_TXI TX interrupt enable

Return value:

- None

__HAL_SMBUS_GET_IT_SOURCE

Description:

- Check whether the specified SMBUS interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __INTERRUPT__: specifies the SMBUS interrupt source to check. This parameter can be one of the following values:
  – SMBUS_IT_ERRI Errors interrupt enable
  – SMBUS_IT_TCI Transfer complete interrupt enable
  – SMBUS_IT_STOPI STOP detection interrupt enable
  – SMBUS_IT_NACKI NACK received interrupt enable
  – SMBUS_IT_ADDRI Address match interrupt enable
  – SMBUS_IT_RXI RX interrupt enable
  – SMBUS_IT_TXI TX interrupt enable

Return value:

- The: new state of __IT__ (SET or RESET).

SMBUS_FLAG_MASK

Description:

- Check whether the specified SMBUS flag is set or not.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_TXIS Transmit interrupt status
  - SMBUS_FLAG_RXNE Receive data register not empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_TC Transfer complete (master mode)
  - SMBUS_FLAG_TCR Transfer complete reload
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert
  - SMBUS_FLAG_BUSY Bus busy
  - SMBUS_FLAG_DIR Transfer direction (slave mode)

Return value:

- The: new state of __FLAG__ (SET or RESET).

__HAL_SMBUS_GET_FLAG

__HAL_SMBUS_CLEAR_FLAG

Description:

- Clear the SMBUS pending flags which are cleared by writing 1 in a specific bit.

Parameters:

- __HANDLE__: specifies the SMBUS Handle.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
  - SMBUS_FLAG_TXE Transmit data register empty
  - SMBUS_FLAG_ADDR Address matched (slave mode)
  - SMBUS_FLAG_AF NACK received flag
  - SMBUS_FLAG_STOPF STOP detection flag
  - SMBUS_FLAG_BERR Bus error
  - SMBUS_FLAG_ARLO Arbitration lost
  - SMBUS_FLAG_OVR Overrun/Underrun
  - SMBUS_FLAG_PECERR PEC error in reception
  - SMBUS_FLAG_TIMEOUT Timeout or Tlow detection flag
  - SMBUS_FLAG_ALERT SMBus alert

Return value:

- None

__HAL_SMBUS_ENABLE

Description:
- Enable the specified SMBUS peripheral.

Parameters:
- __HANDLE__: specifies the SMBUS Handle.

Return value:
- None

__HAL_SMBUS_DISABLE

Description:
- Disable the specified SMBUS peripheral.

Parameters:
- __HANDLE__: specifies the SMBUS Handle.

Return value:
- None

__HAL_SMBUS_GENERATE_NACK

Description:
- Generate a Non-Acknowledge SMBUS peripheral in Slave mode.

Parameters:
- __HANDLE__: specifies the SMBUS Handle.

Return value:
- None

SMBUS Flag definition

SMBUS_FLAG_TXE

SMBUS_FLAG_TXIS

SMBUS_FLAG_RXNE

SMBUS_FLAG_ADDR

SMBUS_FLAG_AF

SMBUS_FLAG_STOPF

SMBUS_FLAG_TC

SMBUS_FLAG_TCR

SMBUS_FLAG_BERR

SMBUS_FLAG_ARLO

SMBUS_FLAG_OVR

SMBUS_FLAG_PECERR

SMBUS_FLAG_TIMEOUT

SMBUS_FLAG_ALERT

SMBUS_FLAG_BUSY

SMBUS_FLAG_DIR

SMBUS general call addressing mode

SMBUS_GENERALCALL_DISABLE

SMBUS_GENERALCALL_ENABLE

SMBUS Interrupt configuration definition

SMBUS_IT_ERRI

SMBUS_IT_TCI

SMBUS_IT_STOPI

SMBUS_IT_NACKI

SMBUS_IT_ADDRI

SMBUS_IT_RXI

SMBUS_IT_TXI

SMBUS_IT_TX

SMBUS_IT_RX

SMBUS_IT_ALERT

SMBUS_IT_ADDR

SMBUS nostretch mode

SMBUS_NOSTRETCH_DISABLE

SMBUS_NOSTRETCH_ENABLE

SMBUS ownaddress2 masks

SMBUS_OA2_NOMASK

SMBUS_OA2_MASK01

SMBUS_OA2_MASK02

SMBUS_OA2_MASK03

SMBUS_OA2_MASK04

SMBUS_OA2_MASK05

SMBUS_OA2_MASK06

SMBUS_OA2_MASK07

SMBUS packet error check mode

SMBUS_PEC_DISABLE

SMBUS_PEC_ENABLE

SMBUS peripheral mode

SMBUS_PERIPHERAL_MODE_SMBUS_HOST

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE

SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE_ARP

SMBUS ReloadEndMode definition

SMBUS_SOFTEND_MODE

SMBUS_RELOAD_MODE

SMBUS_AUTOEND_MODE

SMBUS_SENDPEC_MODE

SMBUS StartStopMode definition

SMBUS_NO_STARTSTOP

SMBUS_GENERATE_NO_START_READ

SMBUS_GENERATE_NO_START_WRITE

SMBUS_GENERATE_STOP

SMBUS_GENERATE_START_READ

SMBUS_GENERATE_START_WRITE

SMBUS XferOptions definition

SMBUS_FIRST_FRAME

SMBUS_NEXT_FRAME

SMBUS_FIRST_AND_LAST_FRAME_NO_PEC

SMBUS_LAST_FRAME_NO_PEC

SMBUS_FIRST_FRAME_WITH_PEC

SMBUS_FIRST_AND_LAST_FRAME_WITH_PEC

SMBUS_LAST_FRAME_WITH_PEC

SMBUS_OTHER_FRAME_NO_PEC

SMBUS_OTHER_FRAME_WITH_PEC

SMBUS_OTHER_AND_LAST_FRAME_NO_PEC

SMBUS_OTHER_AND_LAST_FRAME_WITH_PEC

77 HAL SMBUS Extension Driver

77.1 SMBUSEx Firmware driver registers structures

77.1.1 SMBUS_AutonomousModeConfTypeDef

SMBUS_AutonomousModeConfTypeDef is defined in the stm32u5xx_hal_smbus_ex.h

Data Fields

- ***uint32_t TriggerState***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerPolarity***

Field Documentation

- ***uint32_t SMBUS_AutonomousModeConfTypeDef::TriggerState***
  Specifies the trigger state. This parameter can be a value of ***SMBUSEx_AutonomousMode_FunctionalState***
- ***uint32_t SMBUS_AutonomousModeConfTypeDef::TriggerSelection***
  Specifies the autonomous mode trigger signal selection. This parameter can be a value of ***SMBUSEx_AutonomousMode_TriggerSelection***
- ***uint32_t SMBUS_AutonomousModeConfTypeDef::TriggerPolarity***
  Specifies the autonomous mode trigger signal polarity sensitivity. This parameter can be a value of ***SMBUSEx_AutonomousMode_TriggerPolarity***

77.2 SMBUSEx Firmware driver API description

The following section lists the various functions of the SMBUSEx library.

77.2.1 SMBUS peripheral Extended features

Comparing to other previous devices, the SMBUS interface for STM32U5xx devices contains the following additional features

- Disable or enable wakeup from Stop mode(s)

77.2.2 How to use this driver

77.2.3 WakeUp Mode Functions

This section provides functions allowing to:

- Configure Wake Up Feature

This section contains the following APIs:

- ***HAL_SMBUSEx_EnableWakeUp()***
- ***HAL_SMBUSEx_DisableWakeUp()***

77.2.4 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_SMBUSEx_ConfigFastModePlus()***

77.2.5 Autonomous Mode functions

This section provides functions allowing to:

- Configure Autonomous Mode

This section contains the following APIs:

- *HAL_SMBUSEx_SetConfigAutonomousMode()*
- *HAL_SMBUSEx_GetConfigAutonomousMode()*
- *HAL_SMBUSEx_ClearConfigAutonomousMode()*

77.2.6 Detailed description of functions

HAL_SMBUSEx_EnableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_EnableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

Enable SMBUS wakeup from Stop mode(s).

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_DisableWakeUp

Function name

HAL_StatusTypeDef HAL_SMBUSEx_DisableWakeUp (SMBUS_HandleTypeDef * hsmbus)

Function description

Disable SMBUS wakeup from Stop mode(s).

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_ConfigFastModePlus

Function name

HAL_StatusTypeDef HAL_SMBUSEx_ConfigFastModePlus (SMBUS_HandleTypeDef * hsmbus, uint32_t FastModePlus)

Function description

Configure SMBUS Fast Mode Plus.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
- **FastModePlus:** New state of the Fast Mode Plus.

Return values

- **HAL:** status

HAL_SMBUSEx_SetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_SMBUSEx_SetConfigAutonomousMode (SMBUS_HandleTypeDef * hsmbus, const SMBUS_AutonomousModeConfTypeDef * sConfig)

Function description

Set Autonomous Mode configuration.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
- **sConfig:** Pointer to a SMBUS_AutonomousModeConfTypeDef structure that contains the configuration information of the autonomous mode for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_GetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_SMBUSEx_GetConfigAutonomousMode (const SMBUS_HandleTypeDef * hsmbus, SMBUS_AutonomousModeConfTypeDef * sConfig)

Function description

Get Autonomous Mode configuration.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUSx peripheral.
- **sConfig:** Pointer to a SMBUS_AutonomousModeConfTypeDef structure that contains the configuration information of the autonomous mode for the specified SMBUSx peripheral.

Return values

- **HAL:** status

HAL_SMBUSEx_ClearConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_SMBUSEx_ClearConfigAutonomousMode (SMBUS_HandleTypeDef * hsmbus)

Function description

Clear Autonomous Mode configuration.

Parameters

- **hsmbus:** Pointer to a SMBUS_HandleTypeDef structure that contains the configuration information for the specified SMBUS peripheral.

Return values

- **HAL:** status

77.3 SMBUSEx Firmware driver defines

The following section lists the various define and macros of the module.

77.3.1 SMBUSEx

SMBUSEx

SMBUS Extended Autonomous Mode State

SMBUS_AUTO_MODE_DISABLE

SMBUS_AUTO_MODE_ENABLE

Extended Autonomous Mode Trigger Polarity

SMBUS_TRIG_POLARITY_RISING

SMBUS_TRIG_POLARITY_FALLING

SMBUS Extended Autonomous Mode Trigger Selection

SMBUS_TRIG_GRP1

Trigger Group for I2C1, I2C2, I2C4, I2C5, I2C6 (depends on Product)

SMBUS_TRIG_GRP2

Trigger Group for I2C3

SMBUS_GRP1_GPDMA_CH0_TCF_TRG

HW Trigger signal is GPDMA_CH0_TRG

SMBUS_GRP1_GPDMA_CH1_TCF_TRG

HW Trigger signal is GPDMA_CH1_TRG

SMBUS_GRP1_GPDMA_CH2_TCF_TRG

HW Trigger signal is GPDMA_CH2_TRG

SMBUS_GRP1_GPDMA_CH3_TCF_TRG

HW Trigger signal is GPDMA_CH3_TRG

SMBUS_GRP1_EXTI5_TRG

HW Trigger signal is EXTI5_TRG

SMBUS_GRP1_EXTI9_TRG

HW Trigger signal is EXTI9_TRG

SMBUS_GRP1_LPTIM1_CH1_TRG

HW Trigger signal is LPTIM1_CH1_TRG

SMBUS_GRP1_LPTIM2_CH1_TRG

HW Trigger signal is LPTIM2_CH1_TRG

SMBUS_GRP1_COMP1_TRG

HW Trigger signal is COMP1_TRG

SMBUS_GRP1_COMP2_TRG

HW Trigger signal is COMP2_TRG

SMBUS_GRP1_RTC_ALRA_TRG

HW Trigger signal is RTC_ALRA_TRG

SMBUS_GRP1_RTC_WUT_TRG

HW Trigger signal is RTC_WUT_TRG

SMBUS_GRP2_LPDMA_CH0_TCF_TRG

HW Trigger signal is LPDMA_CH0_TRG

SMBUS_GRP2_LPDMA_CH1_TCF_TRG

HW Trigger signal is LPDMA_CH1_TRG

SMBUS_GRP2_LPDMA_CH2_TCF_TRG

HW Trigger signal is LPDMA_CH2_TRG

SMBUS_GRP2_LPDMA_CH3_TCF_TRG
HW Trigger signal is LPDMA_CH3_TRG

SMBUS_GRP2_EXTI5_TRG
HW Trigger signal is EXTI5_TRG

SMBUS_GRP2_EXTI8_TRG
HW Trigger signal is EXTI8_TRG

SMBUS_GRP2_LPTIM1_CH1_TRG
HW Trigger signal is LPTIM1_CH1_TRG

SMBUS_GRP2_LPTIM3_CH1_TRG
HW Trigger signal is LPTIM3_CH1_TRG

SMBUS_GRP2_COMP1_TRG
HW Trigger signal is COMP1_TRG

SMBUS_GRP2_COMP2_TRG
HW Trigger signal is COMP2_TRG

SMBUS_GRP2_RTC_ALRA_TRG
HW Trigger signal is RTC_ALRA_TRG

SMBUS_GRP2_RTC_WUT_TRG
HW Trigger signal is RTC_WUT_TRG

SMBUS Extended Fast Mode Plus

SMBUS_FASTMODEPLUS_ENABLE
Enable Fast Mode Plus

SMBUS_FASTMODEPLUS_DISABLE
Disable Fast Mode Plus

78 HAL SPI Generic Driver

78.1 SPI Firmware driver registers structures

78.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32u5xx_hal_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Direction***
- ***uint32_t DataSize***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRatePrescaler***
- ***uint32_t FirstBit***
- ***uint32_t TIMode***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPolynomial***
- ***uint32_t CRCLength***
- ***uint32_t NSSPMode***
- ***uint32_t NSSPolarity***
- ***uint32_t FifoThreshold***
- ***uint32_t TxCRCInitializationPattern***
- ***uint32_t RxCRCInitializationPattern***
- ***uint32_t MasterSSIdleness***
- ***uint32_t MasterInterDataIdleness***
- ***uint32_t MasterReceiverAutoSusp***
- ***uint32_t MasterKeepIOState***
- ***uint32_t IOSwap***
- ***uint32_t ReadyMasterManagement***
- ***uint32_t ReadyPolarity***

Field Documentation

- ***uint32_t SPI_InitTypeDef::Mode***
  Specifies the SPI operating mode. This parameter can be a value of ***SPI_Mode***
- ***uint32_t SPI_InitTypeDef::Direction***
  Specifies the SPI bidirectional mode state. This parameter can be a value of ***SPI_Direction***
- ***uint32_t SPI_InitTypeDef::DataSize***
  Specifies the SPI data size. This parameter can be a value of ***SPI_Data_Size***
- ***uint32_t SPI_InitTypeDef::CLKPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_Clock_Polarity***
- ***uint32_t SPI_InitTypeDef::CLKPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_Clock_Phase***
- ***uint32_t SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of ***SPI_Slave_Select_Management***
- ***uint32_t SPI_InitTypeDef::BaudRatePrescaler***
  Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_BaudRate_Prescaler***
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.

- ***uint32_t SPI_InitTypeDef::FirstBit***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_MSB_LSB_Transmission***
- ***uint32_t SPI_InitTypeDef::TIMode***
  Specifies if the TI mode is enabled or not. This parameter can be a value of ***SPI_TI_Mode***
- ***uint32_t SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_CRC_Calculation***
- ***uint32_t SPI_InitTypeDef::CRCPolynomial***
  Specifies the polynomial used for the CRC calculation. This parameter must be an odd number between Min_Data = 0 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. This parameter can be a value of ***SPI_CRC_length***
- ***uint32_t SPI_InitTypeDef::NSSPMode***
  Specifies whether the NSSP signal is enabled or not . This parameter can be a value of ***SPI_NSSP_Mode*** This mode is activated by the SSOM bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0).
- ***uint32_t SPI_InitTypeDef::NSSPolarity***
  Specifies which level of SS input/output external signal (present on SS pin) is considered as active one. This parameter can be a value of ***SPI_NSS_Polarity***
- ***uint32_t SPI_InitTypeDef::FifoThreshold***
  Specifies the FIFO threshold level. This parameter can be a value of ***SPI_Fifo_Threshold***
- ***uint32_t SPI_InitTypeDef::TxCRCInitializationPattern***
  Specifies the transmitter CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::RxCRCInitializationPattern***
  Specifies the receiver CRC initialization Pattern used for the CRC calculation. This parameter can be a value of ***SPI_CRC_Calculation_Initialization_Pattern***
- ***uint32_t SPI_InitTypeDef::MasterSSIdleness***
  Specifies an extra delay, expressed in number of SPI clock cycle periods, inserted additionally between active edge of SS and first data transaction start in master mode. This parameter can be a value of ***SPI_Master_SS_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterInterDataIdleness***
  Specifies minimum time delay (expressed in SPI clock cycles periods) inserted between two consecutive data frames in master mode. This parameter can be a value of ***SPI_Master_InterData_Idleness***
- ***uint32_t SPI_InitTypeDef::MasterReceiverAutoSusp***
  Control continuous SPI transfer in master receiver mode and automatic management in order to avoid overrun condition. This parameter can be a value of ***SPI_Master_RX_AutoSuspend***
- ***uint32_t SPI_InitTypeDef::MasterKeepIOState***
  Control of Alternate function GPIOs state This parameter can be a value of ***SPI_Master_Keep_IO_State***
- ***uint32_t SPI_InitTypeDef::IOSwap***
  Invert MISO/MOSI alternate functions This parameter can be a value of ***SPI_IO_Swap***
- ***uint32_t SPI_InitTypeDef::ReadyMasterManagement***
  Specifies if RDY Signal is managed internally or not. This parameter can be a value of ***SPI_RDY_Master_Management***
- ***uint32_t SPI_InitTypeDef::ReadyPolarity***
  Specifies which level of RDY Signal input (present on RDY pin) is considered as active one. This parameter can be a value of ***SPI_RDY_Polarity***

78.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32u5xx_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***const uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***

- *__IO uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *uint32_t CRCSize*
- *void(* RxISR*
- *void(* TxISR*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_SPI_StateTypeDef State*
- *__IO uint32_t ErrorCode*
- *void(* TxCpltCallback*
- *void(* RxCpltCallback*
- *void(* TxRxCpltCallback*
- *void(* TxHalfCpltCallback*
- *void(* RxHalfCpltCallback*
- *void(* TxRxHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* SuspendCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***SPI_TypeDef* __SPI_HandleTypeDef::Instance***
  SPI registers base address
- ***SPI_InitTypeDef __SPI_HandleTypeDef::Init***
  SPI communication parameters
- ***const uint8_t* __SPI_HandleTypeDef::pTxBuffPtr***
  Pointer to SPI Tx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::TxXferSize***
  SPI Tx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::TxXferCount***
  SPI Tx Transfer Counter
- ***uint8_t* __SPI_HandleTypeDef::pRxBuffPtr***
  Pointer to SPI Rx transfer Buffer
- ***uint16_t __SPI_HandleTypeDef::RxXferSize***
  SPI Rx Transfer size
- ***__IO uint16_t __SPI_HandleTypeDef::RxXferCount***
  SPI Rx Transfer Counter
- ***uint32_t __SPI_HandleTypeDef::CRCSize***
  SPI CRC size used for the transfer
- ***void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Rx ISR
- ***void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)***
  function pointer on Tx ISR
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx***
  SPI Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx***
  SPI Rx DMA Handle parameters
- ***HAL_LockTypeDef __SPI_HandleTypeDef::Lock***
  Locking object

- ***__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State***
  SPI communication state
- ***__IO uint32_t __SPI_HandleTypeDef::ErrorCode***
  SPI Error code
- ***void(* __SPI_HandleTypeDef::TxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Completed callback
- ***void(* __SPI_HandleTypeDef::RxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Completed callback
- ***void(* __SPI_HandleTypeDef::TxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Tx Half Completed callback
- ***void(* __SPI_HandleTypeDef::RxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Rx Half Completed callback
- ***void(* __SPI_HandleTypeDef::TxRxHalfCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI TxRx Half Completed callback
- ***void(* __SPI_HandleTypeDef::ErrorCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Error callback
- ***void(* __SPI_HandleTypeDef::AbortCpltCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Abort callback
- ***void(* __SPI_HandleTypeDef::SuspendCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Suspend callback
- ***void(* __SPI_HandleTypeDef::MspInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp Init callback
- ***void(* __SPI_HandleTypeDef::MspDeInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp DeInit callback

78.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

78.2.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a. Enable the SPIx interface clock
   b. SPI pins configuration
      - Enable the clock for the SPI GPIOs
      - Configure these SPI pins as alternate function push-pull
   c. NVIC configuration if you need to use interrupt process or DMA process
      - Configure the SPIx interrupt priority
      - Enable the NVIC SPI IRQ handle
   d. DMA Configuration if you need to use DMA process
      - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Stream/Channel
      - Enable the DMAx clock
      - Configure the DMA handle parameters
      - Configure the DMA Tx or Rx Stream/Channel
      - Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Stream/Channel
3. Program the Mode, BidirectionalMode , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.

4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Callback registration:

1. The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks: (+) TxCpltCallback : SPI Tx Completed callback (+) RxCpltCallback : SPI Rx Completed callback (+) TxRxCpltCallback : SPI TxRx Completed callback (+) TxHalfCpltCallback : SPI Tx Half Completed callback (+) RxHalfCpltCallback : SPI Rx Half Completed callback (+) TxRxHalfCpltCallback : SPI TxRx Half Completed callback (+) ErrorCallback : SPI Error callback (+) AbortCpltCallback : SPI Abort callback (+) SuspendCallback : SPI Suspend callback (+) MspInitCallback : SPI Msp Init callback (+) MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
2. Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks: (+) TxCpltCallback : SPI Tx Completed callback (+) RxCpltCallback : SPI Rx Completed callback (+) TxRxCpltCallback : SPI TxRx Completed callback (+) TxHalfCpltCallback : SPI Tx Half Completed callback (+) RxHalfCpltCallback : SPI Rx Half Completed callback (+) TxRxHalfCpltCallback : SPI TxRx Half Completed callback (+) ErrorCallback : SPI Error callback (+) AbortCpltCallback : SPI Abort callback (+) SuspendCallback : SPI Suspend callback (+) MspInitCallback : SPI Msp Init callback (+) MspDeInitCallback : SPI Msp DeInit callback By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Callback registration: (#) The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1UL allows the user to configure dynamically the driver callbacks. Use Functions HAL_SPI_RegisterCallback() to register an interrupt callback. Function HAL_SPI_RegisterCallback() allows to register following callbacks:

- TxCpltCallback : SPI Tx Completed callback
- RxCpltCallback : SPI Rx Completed callback
- TxRxCpltCallback : SPI TxRx Completed callback
- TxHalfCpltCallback : SPI Tx Half Completed callback
- RxHalfCpltCallback : SPI Rx Half Completed callback
- TxRxHalfCpltCallback : SPI TxRx Half Completed callback
- ErrorCallback : SPI Error callback
- AbortCpltCallback : SPI Abort callback
- SuspendCallback : SPI Suspend callback
- MspInitCallback : SPI Msp Init callback
- MspDeInitCallback : SPI Msp DeInit callback This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. (#) Use function HAL_SPI_UnRegisterCallback to reset a callback to the default weak function. HAL_SPI_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- TxCpltCallback : SPI Tx Completed callback
- RxCpltCallback : SPI Rx Completed callback
- TxRxCpltCallback : SPI TxRx Completed callback
- TxHalfCpltCallback : SPI Tx Half Completed callback
- RxHalfCpltCallback : SPI Rx Half Completed callback

- TxRxHalfCpltCallback : SPI TxRx Half Completed callback
- ErrorCallback : SPI Error callback
- AbortCpltCallback : SPI Abort callback
- SuspendCallback : SPI Suspend callback
- MspInitCallback : SPI Msp Init callback
- MspDeInitCallback : SPI Msp DeInit callback By default, after the HAL_SPI_Init() and when the state is HAL_SPI_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_SPI_MasterTxCpltCallback(), HAL_SPI_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_SPI_Init()/ HAL_SPI_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_SPI_Init()/ HAL_SPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state. Callbacks can be registered/unregistered in HAL_SPI_STATE_READY state only. Exception done MspInit/MspDeInit functions that can be registered/unregistered in HAL_SPI_STATE_READY or HAL_SPI_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_SPI_RegisterCallback() before calling HAL_SPI_DeInit() or HAL_SPI_Init() function. When The compilation define USE_HAL_PPP_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak callbacks are used. SuspendCallback restriction: SuspendCallback is called only when MasterReceiverAutoSusp is enabled and EOT interrupt is activated. SuspendCallback is used in relation with functions HAL_SPI_Transmit_IT, HAL_SPI_Receive_IT and HAL_SPI_TransmitReceive_IT.

Circular mode restriction:

- The DMA circular mode cannot be used when the SPI is configured in these modes:
  - Master 2Lines RxOnly
  - Master 1Line Rx
- The CRC feature is not managed when the DMA circular mode is enabled
- The functions HAL_SPI_DMAPause()/ HAL_SPI_DMAResume() are not supported. Return always HAL_ERROR with ErrorCode set to HAL_SPI_ERROR_NOT_SUPPORTED. Those functions are maintained for backward compatibility reasons.

78.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
  - Mode
  - Direction
  - Data Size
  - Clock Polarity and Phase
  - NSS Management
  - BaudRate Prescaler
  - FirstBit
  - TIMode
  - CRC Calculation
  - CRC Polynomial if CRC enabled
  - CRC Length, used only with Data8 and Data16
  - FIFO reception threshold
  - FIFO transmission threshold
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- ***HAL_SPI_Init()***
- ***HAL_SPI_DeInit()***
- ***HAL_SPI_MspInit()***

- *HAL_SPI_MspDeInit()*
- *HAL_SPI_RegisterCallback()*
- *HAL_SPI_UnRegisterCallback()*

78.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the SPI data transfers.

The SPI supports master and slave mode :

1. There are two modes of transfer:
   a. Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   b. No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback()user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- *HAL_SPI_Transmit()*
- *HAL_SPI_Receive()*
- *HAL_SPI_TransmitReceive()*
- *HAL_SPI_Transmit_IT()*
- *HAL_SPI_Receive_IT()*
- *HAL_SPI_TransmitReceive_IT()*
- *HAL_SPI_Transmit_DMA()*
- *HAL_SPI_Receive_DMA()*
- *HAL_SPI_TransmitReceive_DMA()*
- *HAL_SPI_Abort()*
- *HAL_SPI_Abort_IT()*
- *HAL_SPI_DMAPause()*
- *HAL_SPI_DMAResume()*
- *HAL_SPI_DMAStop()*
- *HAL_SPI_IRQHandler()*
- *HAL_SPI_TxCpltCallback()*
- *HAL_SPI_RxCpltCallback()*
- *HAL_SPI_TxRxCpltCallback()*
- *HAL_SPI_TxHalfCpltCallback()*
- *HAL_SPI_RxHalfCpltCallback()*
- *HAL_SPI_TxRxHalfCpltCallback()*
- *HAL_SPI_ErrorCallback()*
- *HAL_SPI_AbortCpltCallback()*
- *HAL_SPI_SuspendCallback()*

78.2.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- HAL_SPI_GetState() API can be helpful to check in run-time the state of the SPI peripheral
- HAL_SPI_GetError() check in run-time Errors occurring during communication

This section contains the following APIs:

- *HAL_SPI_GetState()*
- *HAL_SPI_GetError()*

78.2.5 Detailed description of functions

HAL_SPI_Init

Function name

HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI according to the specified parameters in the SPI_InitTypeDef and initialize the associated handle.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_DeInit

Function name

HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI peripheral.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **HAL:** status

HAL_SPI_MspInit

Function name

void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)

Function description

Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_MspDeInit

Function name

void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)

Function description

De-Initialize the SPI MSP.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_RegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID, pSPI_CallbackTypeDef pCallback)

Function description

Register a User SPI Callback To be used instead of the weak predefined callback.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI.
- **CallbackID:** ID of the callback to be registered
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_SPI_RegisterCallback() may be called before HAL_SPI_Init() in HAL_SPI_STATE_RESET to register callbacks for HAL_SPI_MSPINIT_CB_ID and HAL_SPI_MSPDEINIT_CB_ID

HAL_SPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SPI_UnRegisterCallback (SPI_HandleTypeDef * hspi, HAL_SPI_CallbackIDTypeDef CallbackID)

Function description

Unregister an SPI Callback SPI callback is redirected to the weak predefined callback.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI.
- **CallbackID:** ID of the callback to be unregistered

Return values

- **HAL:** status

Notes

- The HAL_SPI_UnRegisterCallback() may be called before HAL_SPI_Init() in HAL_SPI_STATE_RESET to un-register callbacks for HAL_SPI_MSPINIT_CB_ID and HAL_SPI_MSPDEINIT_CB_ID

HAL_SPI_Transmit

Function name

HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** : pointer to data buffer
- **Size:** : amount of data to be sent
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Receive

Function name

HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** : pointer to data buffer
- **Size:** : amount of data to be received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_TransmitReceive

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Transmit and Receive an amount of data in blocking mode.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent and received
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_SPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent and received

Return values

- **HAL:** status

HAL_SPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

HAL_SPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pData Length must be Size + 1.

HAL_SPI_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Transmit and Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hspi:** : pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer
- **Size:** : amount of data to be sent

Return values

- **HAL:** status

Notes

- When the CRC feature is enabled the pRxData Length must be Size + 1

HAL_SPI_DMAPause

Function name

HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)

Function description

Pause the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAResume

Function name

HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)

Function description

Resume the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_DMAStop

Function name

HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)

Function description

Stop the DMA Transfer.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL_ERROR:**

HAL_SPI_Abort

Function name

HAL_StatusTypeDef HAL_SPI_Abort (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (blocking mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode) + Set handle State to READY.
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_SPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_SPI_Abort_IT (SPI_HandleTypeDef * hspi)

Function description

Abort ongoing transfer (Interrupt mode).

Parameters

- **hspi:** SPI handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer (Tx and Rx), started in Interrupt or DMA mode.
- This procedure performs following operations : + Disable SPI Interrupts (depending of transfer direction) + Disable the DMA transfer in the peripheral register (if enabled) + Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode) + Set handle State to READY + At abort completion, call user abort complete callback.
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_SPI_IRQHandler

Function name

void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)

Function description

Handle SPI interrupt request.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **None:**

HAL_SPI_TxCpltCallback

Function name

void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxCpltCallback

Function name

void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxCpltCallback

Function name

void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxHalfCpltCallback

Function name

void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_RxHalfCpltCallback

Function name

void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_TxRxHalfCpltCallback

Function name

void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)

Function description

Tx and Rx Half Transfer callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_ErrorCallback

Function name

void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)

Function description

SPI error callback.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPI_AbortCpltCallback

Function name

void HAL_SPI_AbortCpltCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Abort Complete callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_SuspendCallback

Function name

void HAL_SPI_SuspendCallback (SPI_HandleTypeDef * hspi)

Function description

SPI Suspend callback.

Parameters

- **hspi:** SPI handle.

Return values

- **None:**

HAL_SPI_GetState

Function name

HAL_SPI_StateTypeDef HAL_SPI_GetState (const SPI_HandleTypeDef * hspi)

Function description

Return the SPI handle state.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** state

HAL_SPI_GetError

Function name

uint32_t HAL_SPI_GetError (const SPI_HandleTypeDef * hspi)

Function description

Return the SPI error code.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **SPI:** error code in bitmap format

78.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

78.3.1 SPI

SPI

Autonomous Mode Trigger Polarity

SPI_TRIG_POLARITY_RISING

SPI_TRIG_POLARITY_FALLING

Autonomous Mode Trigger Selection

SPI_TRIG_GRP1

SPI_TRIG_GRP2

SPI_GRP1_GPDMA_CH0_TCF_TRG

SPI_GRP1_GPDMA_CH1_TCF_TRG

SPI_GRP1_GPDMA_CH2_TCF_TRG

SPI_GRP1_GPDMA_CH3_TCF_TRG

SPI_GRP1_EXTI4_TRG

SPI_GRP1_EXTI9_TRG

SPI_GRP1_LPTIM1_CH1_TRG

SPI_GRP1_LPTIM2_CH1_TRG

SPI_GRP1_COMP1_TRG

SPI_GRP1_COMP2_TRG

SPI_GRP1_RTC_ALRA_TRG

SPI_GRP1_RTC_WUT_TRG

SPI_GRP2_LPDMA_CH0_TCF_TRG

SPI_GRP2_LPDMA_CH1_TCF_TRG

SPI_GRP2_LPDMA_CH2_TCF_TRG

SPI_GRP2_LPDMA_CH3_TCF_TRG

SPI_GRP2_EXTI4_TRG

SPI_GRP2_EXTI8_TRG

SPI_GRP2_LPTIM1_CH1_TRG

SPI_GRP2_LPTIM3_CH1_TRG

SPI_GRP2_COMP1_TRG

SPI_GRP2_COMP2_TRG

SPI_GRP2_RTC_ALRA_TRG

SPI_GRP2_RTC_WUT_TRG

SPI BaudRate Prescaler

SPI_BAUDRATEPRESCALER_BYPASS

SPI_BAUDRATEPRESCALER_2

SPI_BAUDRATEPRESCALER_4

SPI_BAUDRATEPRESCALER_8

SPI_BAUDRATEPRESCALER_16

SPI_BAUDRATEPRESCALER_32

SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

SPI CRC Calculation Initialization Pattern

SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN

SPI_CRC_INITIALIZATION_ALL_ONE_PATTERN

SPI CRC Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_4BIT

SPI_CRC_LENGTH_5BIT

SPI_CRC_LENGTH_6BIT

SPI_CRC_LENGTH_7BIT

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_9BIT

SPI_CRC_LENGTH_10BIT

SPI_CRC_LENGTH_11BIT

SPI_CRC_LENGTH_12BIT

SPI_CRC_LENGTH_13BIT

SPI_CRC_LENGTH_14BIT

SPI_CRC_LENGTH_15BIT

SPI_CRC_LENGTH_16BIT

SPI_CRC_LENGTH_17BIT

SPI_CRC_LENGTH_18BIT

SPI_CRC_LENGTH_19BIT

SPI_CRC_LENGTH_20BIT

SPI_CRC_LENGTH_21BIT

SPI_CRC_LENGTH_22BIT

SPI_CRC_LENGTH_23BIT

SPI_CRC_LENGTH_24BIT

SPI_CRC_LENGTH_25BIT

SPI_CRC_LENGTH_26BIT

SPI_CRC_LENGTH_27BIT

SPI_CRC_LENGTH_28BIT

SPI_CRC_LENGTH_29BIT

SPI_CRC_LENGTH_30BIT

SPI_CRC_LENGTH_31BIT

SPI_CRC_LENGTH_32BIT

SPI Data Size

SPI_DATASIZE_4BIT

SPI_DATASIZE_5BIT

SPI_DATASIZE_6BIT

SPI_DATASIZE_7BIT

SPI_DATASIZE_8BIT

SPI_DATASIZE_9BIT

SPI_DATASIZE_10BIT

SPI_DATASIZE_11BIT

SPI_DATASIZE_12BIT

SPI_DATASIZE_13BIT

SPI_DATASIZE_14BIT

SPI_DATASIZE_15BIT

SPI_DATASIZE_16BIT

SPI_DATASIZE_17BIT

SPI_DATASIZE_18BIT

SPI_DATASIZE_19BIT

SPI_DATASIZE_20BIT

SPI_DATASIZE_21BIT

SPI_DATASIZE_22BIT

SPI_DATASIZE_23BIT

SPI_DATASIZE_24BIT

SPI_DATASIZE_25BIT

SPI_DATASIZE_26BIT

SPI_DATASIZE_27BIT

SPI_DATASIZE_28BIT

SPI_DATASIZE_29BIT

SPI_DATASIZE_30BIT

SPI_DATASIZE_31BIT

SPI_DATASIZE_32BIT

SPI Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_TXONLY

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Codes

HAL_SPI_ERROR_NONE

No error

HAL_SPI_ERROR_MODF

MODF error

HAL_SPI_ERROR_CRC

CRC error

HAL_SPI_ERROR_OVR

OVR error

HAL_SPI_ERROR_FRE

FRE error

HAL_SPI_ERROR_DMA

DMA transfer error

HAL_SPI_ERROR_FLAG

Error on RXP/TXP/DXP/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT

Error during SPI Abort procedure

HAL_SPI_ERROR_UDR

Underrun error

HAL_SPI_ERROR_TIMEOUT

Timeout error

HAL_SPI_ERROR_UNKNOW

Unknown error

HAL_SPI_ERROR_NOT_SUPPORTED

Requested operation not supported

HAL_SPI_ERROR_INVALID_CALLBACK

Invalid Callback error

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE

Description:

- Reset SPI handle state.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT

Description:

- Enable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  – SPI_IT_RXP : Rx-Packet available interrupt
  – SPI_IT_TXP : Tx-Packet space available interrupt
  – SPI_IT_DXP : Duplex Packet interrupt
  – SPI_IT_EOT : End of transfer interrupt
  – SPI_IT_TXTF : Transmission Transfer Filled interrupt
  – SPI_IT_UDR : Underrun interrupt
  – SPI_IT_OVR : Overrun interrupt
  – SPI_IT_CRCERR : CRC error interrupt
  – SPI_IT_FRE : TI mode frame format error interrupt
  – SPI_IT_MODF : Mode fault interrupt
  – SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  – SPI_IT_RXP : Rx-Packet available interrupt
  – SPI_IT_TXP : Tx-Packet space available interrupt
  – SPI_IT_DXP : Duplex Packet interrupt
  – SPI_IT_EOT : End of transfer interrupt
  – SPI_IT_TXTF : Transmission Transfer Filled interrupt
  – SPI_IT_UDR : Underrun interrupt
  – SPI_IT_OVR : Overrun interrupt
  – SPI_IT_CRCERR : CRC error interrupt
  – SPI_IT_FRE : TI mode frame format error interrupt
  – SPI_IT_MODF : Mode fault interrupt
  – SPI_IT_ERR : Error interrupt

Return value:

- None

__HAL_SPI_GET_IT_SOURCE

Description:

- Check whether the specified SPI interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_RXP : Rx-Packet available interrupt
  - SPI_IT_TXP : Tx-Packet space available interrupt
  - SPI_IT_DXP : Duplex Packet interrupt
  - SPI_IT_EOT : End of transfer interrupt
  - SPI_IT_TXTF : Transmission Transfer Filled interrupt
  - SPI_IT_UDR : Underrun interrupt
  - SPI_IT_OVR : Overrun interrupt
  - SPI_IT_CRCERR : CRC error interrupt
  - SPI_IT_FRE : TI mode frame format error interrupt
  - SPI_IT_MODF : Mode fault interrupt
  - SPI_IT_ERR : Error interrupt

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SPI_GET_FLAG

Description:

- Check whether the specified SPI flag is set or not.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, 3, 4, 5 or 6 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXP : Rx-Packet available flag
  - SPI_FLAG_TXP : Tx-Packet space available flag
  - SPI_FLAG_DXP : Duplex Packet flag
  - SPI_FLAG_EOT : End of transfer flag
  - SPI_FLAG_TXTF : Transmission Transfer Filled flag
  - SPI_FLAG_UDR : Underrun flag
  - SPI_FLAG_OVR : Overrun flag
  - SPI_FLAG_CRCERR : CRC error flag
  - SPI_FLAG_FRE : TI mode frame format error flag
  - SPI_FLAG_MODF : Mode fault flag
  - SPI_FLAG_SUSP : Transfer suspend complete flag
  - SPI_FLAG_TXC : TxFIFO transmission complete flag
  - SPI_FLAG_FRLVL : Fifo reception level flag
  - SPI_FLAG_RXWNE : RxFIFO word not empty flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_SPI_CLEAR_CRCERRFLAG

Description:

- Clear the SPI CRCERR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_UDRFLAG

Description:

- Clear the SPI UDR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_EOTFLAG

Description:

- Clear the SPI EOT pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle.

Return value:

- None

__HAL_SPI_CLEAR_TXTFFLAG

Description:
- Clear the SPI UDR pending flag.

Parameters:
- __HANDLE__: specifies the SPI Handle.

Return value:
- None

__HAL_SPI_CLEAR_SUSPFLAG

Description:
- Clear the SPI SUSP pending flag.

Parameters:
- __HANDLE__: specifies the SPI Handle.

Return value:
- None

__HAL_SPI_ENABLE

Description:
- Enable the SPI peripheral.

Parameters:
- __HANDLE__: specifies the SPI Handle.

Return value:
- None

__HAL_SPI_DISABLE

Description:
- Disable the SPI peripheral.

Parameters:
- __HANDLE__: specifies the SPI Handle.

Return value:
- None

SPI Fifo Threshold

SPI_FIFO_THRESHOLD_01DATA

SPI_FIFO_THRESHOLD_02DATA

SPI_FIFO_THRESHOLD_03DATA

SPI_FIFO_THRESHOLD_04DATA

SPI_FIFO_THRESHOLD_05DATA

SPI_FIFO_THRESHOLD_06DATA

SPI_FIFO_THRESHOLD_07DATA

SPI_FIFO_THRESHOLD_08DATA

SPI_FIFO_THRESHOLD_09DATA

SPI_FIFO_THRESHOLD_10DATA

SPI_FIFO_THRESHOLD_11DATA

SPI_FIFO_THRESHOLD_12DATA

SPI_FIFO_THRESHOLD_13DATA

SPI_FIFO_THRESHOLD_14DATA

SPI_FIFO_THRESHOLD_15DATA

SPI_FIFO_THRESHOLD_16DATA

SPI FIFO Type

SPI_LOWEND_FIFO_SIZE

SPI_HIGHEND_FIFO_SIZE

SPI Flags Definition

SPI_FLAG_RXP

SPI_FLAG_TXP

SPI_FLAG_DXP

SPI_FLAG_EOT

SPI_FLAG_TXTF

SPI_FLAG_UDR

SPI_FLAG_OVR

SPI_FLAG_CRCERR

SPI_FLAG_FRE

SPI_FLAG_MODF

SPI_FLAG_SUSP

SPI_FLAG_TXC

SPI_FLAG_FRLVL

SPI_FLAG_RXWNE

SPI Interrupt Definition

SPI_IT_RXP

SPI_IT_TXP

SPI_IT_DXP

SPI_IT_EOT

SPI_IT_TXTF

SPI_IT_UDR

SPI_IT_OVR

SPI_IT_CRCERR

SPI_IT_FRE

SPI_IT_MODF

SPI_IT_ERR

Control SPI IO Swap

SPI_IO_SWAP_DISABLE

SPI_IO_SWAP_ENABLE

SPI Master Inter-Data Idleness

SPI_MASTER_INTERDATA_IDLENESS_00CYCLE

SPI_MASTER_INTERDATA_IDLENESS_01CYCLE

SPI_MASTER_INTERDATA_IDLENESS_02CYCLE

SPI_MASTER_INTERDATA_IDLENESS_03CYCLE

SPI_MASTER_INTERDATA_IDLENESS_04CYCLE

SPI_MASTER_INTERDATA_IDLENESS_05CYCLE

SPI_MASTER_INTERDATA_IDLENESS_06CYCLE

SPI_MASTER_INTERDATA_IDLENESS_07CYCLE

SPI_MASTER_INTERDATA_IDLENESS_08CYCLE

SPI_MASTER_INTERDATA_IDLENESS_09CYCLE

SPI_MASTER_INTERDATA_IDLENESS_10CYCLE

SPI_MASTER_INTERDATA_IDLENESS_11CYCLE

SPI_MASTER_INTERDATA_IDLENESS_12CYCLE

SPI_MASTER_INTERDATA_IDLENESS_13CYCLE

SPI_MASTER_INTERDATA_IDLENESS_14CYCLE

SPI_MASTER_INTERDATA_IDLENESS_15CYCLE

Keep IO State

SPI_MASTER_KEEP_IO_STATE_DISABLE

SPI_MASTER_KEEP_IO_STATE_ENABLE

SPI Master Receiver AutoSuspend

SPI_MASTER_RX_AUTOSUSP_DISABLE

SPI_MASTER_RX_AUTOSUSP_ENABLE

SPI Master SS Idleness

SPI_MASTER_SS_IDLENESS_00CYCLE

SPI_MASTER_SS_IDLENESS_01CYCLE

SPI_MASTER_SS_IDLENESS_02CYCLE

SPI_MASTER_SS_IDLENESS_03CYCLE

SPI_MASTER_SS_IDLENESS_04CYCLE

SPI_MASTER_SS_IDLENESS_05CYCLE

SPI_MASTER_SS_IDLENESS_06CYCLE

SPI_MASTER_SS_IDLENESS_07CYCLE

SPI_MASTER_SS_IDLENESS_08CYCLE

SPI_MASTER_SS_IDLENESS_09CYCLE

SPI_MASTER_SS_IDLENESS_10CYCLE

SPI_MASTER_SS_IDLENESS_11CYCLE

SPI_MASTER_SS_IDLENESS_12CYCLE

SPI_MASTER_SS_IDLENESS_13CYCLE

SPI_MASTER_SS_IDLENESS_14CYCLE

SPI_MASTER_SS_IDLENESS_15CYCLE

SPI Mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

SPI MSB LSB Transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

SPI NSS Pulse Mode

SPI_NSS_PULSE_DISABLE

SPI_NSS_PULSE_ENABLE

SPI NSS Polarity

SPI_NSS_POLARITY_LOW

SPI_NSS_POLARITY_HIGH

SPI RDY Signal Input Master Management

SPI_RDY_MASTER_MANAGEMENT_INTERNALLY

SPI_RDY_MASTER_MANAGEMENT_EXTERNALLY

SPI RDY Signal Input/Output Polarity

SPI_RDY_POLARITY_HIGH

SPI_RDY_POLARITY_LOW

SPI Reception FIFO Status Level

SPI_RX_FIFO_0PACKET

SPI_RX_FIFO_1PACKET

SPI_RX_FIFO_2PACKET

SPI_RX_FIFO_3PACKET

SPI Slave Select Management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

SPI TI Mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

SPI Underrun Behavior

SPI_UNDERRUN_BEHAV_REGISTER_PATTERN

SPI_UNDERRUN_BEHAV_LAST_RECEIVED

79 HAL SPI Extension Driver

79.1 SPIEx Firmware driver registers structures

79.1.1 SPI_AutonomousModeConfTypeDef

SPI_AutonomousModeConfTypeDef is defined in the stm32u5xx_hal_spi_ex.h

Data Fields

- ***uint32_t TriggerState***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerPolarity***

Field Documentation

- ***uint32_t SPI_AutonomousModeConfTypeDef::TriggerState***
  Specifies the trigger state. This parameter can be a value of ***FunctionalState***
- ***uint32_t SPI_AutonomousModeConfTypeDef::TriggerSelection***
  Specifies the autonomous mode trigger signal selection. This parameter can be a value of ***SPI_AutonomousMode_TriggerSelection***
- ***uint32_t SPI_AutonomousModeConfTypeDef::TriggerPolarity***
  Specifies the autonomous mode trigger signal polarity sensitivity. This parameter can be a value of ***SPI_AutonomousMode_TriggerPolarity***

79.2 SPIEx Firmware driver API description

The following section lists the various functions of the SPIEx library.

79.2.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. SPIEx function:
   – HAL_SPIEx_FlushRxFifo()
   – HAL_SPIEx_FlushRxFifo()
   – HAL_SPIEx_EnableLockConfiguration()
   – HAL_SPIEx_ConfigureUnderrun()

This section contains the following APIs:

- ***HAL_SPIEx_FlushRxFifo()***
- ***HAL_SPIEx_EnableLockConfiguration()***
- ***HAL_SPIEx_ConfigureUnderrun()***
- ***HAL_SPIEx_SetConfigAutonomousMode()***
- ***HAL_SPIEx_GetConfigAutonomousMode()***
- ***HAL_SPIEx_ClearConfigAutonomousMode()***

79.2.2 Detailed description of functions

HAL_SPIEx_FlushRxFifo

Function name

HAL_StatusTypeDef HAL_SPIEx_FlushRxFifo (const SPI_HandleTypeDef * hspi)

Function description

Flush the RX fifo.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.

Return values

- **HAL:** status

HAL_SPIEx_EnableLockConfiguration

Function name

HAL_StatusTypeDef HAL_SPIEx_EnableLockConfiguration (SPI_HandleTypeDef * hspi)

Function description

Enable the Lock for the AF configuration of associated IOs and write protect the Content of Configuration register 2 when SPI is enabled.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.

Return values

- **None:**

HAL_SPIEx_ConfigureUnderrun

Function name

HAL_StatusTypeDef HAL_SPIEx_ConfigureUnderrun (SPI_HandleTypeDef * hspi, uint32_t UnderrunDetection, uint32_t UnderrunBehaviour)

Function description

Configure the UNDERRUN condition and behavior of slave transmitter.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **UnderrunDetection:** : Detection of underrun condition at slave transmitter This parameter is not supported in this SPI version. It is kept in order to not break the compatibility.
- **UnderrunBehaviour:** : Behavior of slave transmitter at underrun condition This parameter can be a value of SPI Underrun Behavior.

Return values

- **None:**

HAL_SPIEx_SetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_SPIEx_SetConfigAutonomousMode (SPI_HandleTypeDef * hspi, const SPI_AutonomousModeConfTypeDef * sConfig)

Function description

Set Autonomous Mode configuration.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPIx peripheral.
- **sConfig:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information of the autonomous mode for the specified SPIx peripheral.

Return values

- **HAL:** status

HAL_SPIEx_GetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_SPIEx_GetConfigAutonomousMode (const SPI_HandleTypeDef * hspi, SPI_AutonomousModeConfTypeDef * sConfig)

Function description

Get Autonomous Mode configuration.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPIx peripheral.
- **sConfig:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information of the autonomous mode for the specified SPIx peripheral.

Return values

- **HAL:** status

HAL_SPIEx_ClearConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_SPIEx_ClearConfigAutonomousMode (SPI_HandleTypeDef * hspi)

Function description

Clear Autonomous Mode configuration.

Parameters

- **hspi:** Pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPIx peripheral.

Return values

- **HAL:** status

79.3 SPIEx Firmware driver defines

The following section lists the various define and macros of the module.

79.3.1 SPIEx

SPIEx

SPIEx Extended Exported Macros

IS_SPI_AUTO_MODE

IS_SPI_AUTONOMOUS_INSTANCE

IS_SPI_TRIG_SOURCE

IS_SPI_GRP1_TRIG_SOURCE

IS_SPI_GRP2_TRIG_SOURCE

IS_SPI_AUTO_MODE_TRG_POL

80 HAL SRAM Generic Driver

80.1 SRAM Firmware driver registers structures

80.1.1 __SRAM_HandleTypeDef

__SRAM_HandleTypeDef is defined in the stm32u5xx_hal_sram.h

Data Fields

- ***FMC_NORSRAM_TypeDef * Instance***
- ***FMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SRAM_StateTypeDef State***
- ***DMA_HandleTypeDef * hdma***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***
- ***void(* DmaXferCpltCallback***
- ***void(* DmaXferErrorCallback***

Field Documentation

- ***FMC_NORSRAM_TypeDef* __SRAM_HandleTypeDef::Instance***
  Register base address
- ***FMC_NORSRAM_EXTENDED_TypeDef* __SRAM_HandleTypeDef::Extended***
  Extended mode register base address
- ***FMC_NORSRAM_InitTypeDef __SRAM_HandleTypeDef::Init***
  SRAM device control configuration parameters
- ***HAL_LockTypeDef __SRAM_HandleTypeDef::Lock***
  SRAM locking object
- ***__IO HAL_SRAM_StateTypeDef __SRAM_HandleTypeDef::State***
  SRAM device access state
- ***DMA_HandleTypeDef* __SRAM_HandleTypeDef::hdma***
  Pointer DMA handler
- ***void(* __SRAM_HandleTypeDef::MspInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp Init callback
- ***void(* __SRAM_HandleTypeDef::MspDeInitCallback)(struct __SRAM_HandleTypeDef *hsram)***
  SRAM Msp DeInit callback
- ***void(* __SRAM_HandleTypeDef::DmaXferCpltCallback)(DMA_HandleTypeDef *hdma)***
  SRAM DMA Xfer Complete callback
- ***void(* __SRAM_HandleTypeDef::DmaXferErrorCallback)(DMA_HandleTypeDef *hdma)***
  SRAM DMA Xfer Error callback

80.2 SRAM Firmware driver API description

The following section lists the various functions of the SRAM library.

80.2.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example: SRAM_HandleTypeDef hsram; and:
   – Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
   – Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
   – Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode
2. Declare two FMC_NORSRAM_TimingTypeDef structures, for both normal and extended mode timings; for example: FMC_NORSRAM_TimingTypeDef Timing and FMC_NORSRAM_TimingTypeDef ExTiming; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function HAL_SRAM_Init(). This function performs the following sequence:
   a. MSP hardware layer configuration using the function HAL_SRAM_MspInit()
   b. Control register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Init()
   c. Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Timing_Init()
   d. Extended mode Timing register configuration using the FMC NORSRAM interface function FMC_NORSRAM_Extended_Timing_Init()
   e. Enable the SRAM device using the macro __FMC_NORSRAM_ENABLE()
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
   – HAL_SRAM_Read()/HAL_SRAM_Write() for polling read/write access
   – HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA() for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs HAL_SRAM_WriteOperation_Enable()/ HAL_SRAM_WriteOperation_Disable() to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function HAL_SRAM_GetState()

Callback registration

The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_SRAM_RegisterCallback() to register a user callback, it allows to register following callbacks:

- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_SRAM_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:
- MspInitCallback : SRAM MspInit.
- MspDeInitCallback : SRAM MspDeInit. This function) takes as parameters the HAL peripheral handle and the Callback ID. By default, after the HAL_SRAM_Init and if the state is HAL_SRAM_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_SRAM_Init and HAL_SRAM_DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_SRAM_Init and HAL_SRAM_DeInit keep and use the user MspInit/MspDeInit callbacks (registered beforehand) Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_SRAM_RegisterCallback before calling HAL_SRAM_DeInit or HAL_SRAM_Init function. When The compilation define USE_HAL_SRAM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

80.2.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- ***HAL_SRAM_Init()***
- ***HAL_SRAM_DeInit()***
- ***HAL_SRAM_MspInit()***

- *HAL_SRAM_MspDeInit()*
- *HAL_SRAM_DMA_XferCpltCallback()*
- *HAL_SRAM_DMA_XferErrorCallback()*

80.2.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory
This section contains the following APIs:
- *HAL_SRAM_Read_8b()*
- *HAL_SRAM_Write_8b()*
- *HAL_SRAM_Read_16b()*
- *HAL_SRAM_Write_16b()*
- *HAL_SRAM_Read_32b()*
- *HAL_SRAM_Write_32b()*
- *HAL_SRAM_Read_DMA()*
- *HAL_SRAM_Write_DMA()*
- *HAL_SRAM_RegisterCallback()*
- *HAL_SRAM_UnRegisterCallback()*
- *HAL_SRAM_RegisterDmaCallback()*
- *HAL_SRAM_DMA_XferCpltCallback()*
- *HAL_SRAM_DMA_XferErrorCallback()*

80.2.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.
This section contains the following APIs:
- *HAL_SRAM_WriteOperation_Enable()*
- *HAL_SRAM_WriteOperation_Disable()*

80.2.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.
This section contains the following APIs:
- *HAL_SRAM_GetState()*

80.2.6 Detailed description of functions

HAL_SRAM_Init

Function name

HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FMC_NORSRAM_TimingTypeDef * Timing, FMC_NORSRAM_TimingTypeDef * ExtTiming)

Function description

Performs the SRAM device initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **Timing:** Pointer to SRAM control timing structure
- **ExtTiming:** Pointer to SRAM extended mode timing structure

Return values

- **HAL:** status

HAL_SRAM_DeInit

Function name

HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)

Function description

Performs the SRAM device De-initialization sequence.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_MspInit

Function name

void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP Init.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_MspDeInit

Function name

void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)

Function description

SRAM MSP DeInit.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_Read_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 8-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_8b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 8-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_16b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Read_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_32b

Function name

HAL_StatusTypeDef HAL_SRAM_Write_32b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_Read_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)

Function description

Reads a Words data from the SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to read start address
- **pDstBuffer:** Pointer to destination buffer
- **BufferSize:** Size of the buffer to read from memory

Return values

- **HAL:** status

HAL_SRAM_Write_DMA

Function name

HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)

Function description

Writes a Words data buffer to SRAM memory using DMA transfer.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
- **pAddress:** Pointer to write start address
- **pSrcBuffer:** Pointer to source buffer to write
- **BufferSize:** Size of the buffer to write to memory

Return values

- **HAL:** status

HAL_SRAM_DMA_XferCpltCallback

Function name

void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete callback.

Parameters

- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_DMA_XferErrorCallback

Function name

void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)

Function description

DMA transfer complete error callback.

Parameters

- **hdma:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **None:**

HAL_SRAM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_CallbackTypeDef pCallback)

Function description

Register a User SRAM Callback To be used to override the weak predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SRAM_MSP_INIT_CB_ID SRAM MspInit callback ID
  - HAL_SRAM_MSP_DEINIT_CB_ID SRAM MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SRAM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_SRAM_UnRegisterCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId)

Function description

Unregister a User SRAM Callback SRAM Callback is redirected to the weak predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_SRAM_MSP_INIT_CB_ID SRAM MspInit callback ID
  - HAL_SRAM_MSP_DEINIT_CB_ID SRAM MspDeInit callback ID
  - HAL_SRAM_DMA_XFER_CPLT_CB_ID SRAM DMA Xfer Complete callback ID
  - HAL_SRAM_DMA_XFER_ERR_CB_ID SRAM DMA Xfer Error callback ID

Return values

- **status:**

HAL_SRAM_RegisterDmaCallback

Function name

HAL_StatusTypeDef HAL_SRAM_RegisterDmaCallback (SRAM_HandleTypeDef * hsram, HAL_SRAM_CallbackIDTypeDef CallbackId, pSRAM_DmaCallbackTypeDef pCallback)

Function description

Register a User SRAM Callback for DMA transfers To be used to override the weak predefined callback.

Parameters

- **hsram:** : SRAM handle
- **CallbackId:** : ID of the callback to be registered This parameter can be one of the following values:
  - HAL_SRAM_DMA_XFER_CPLT_CB_ID SRAM DMA Xfer Complete callback ID
  - HAL_SRAM_DMA_XFER_ERR_CB_ID SRAM DMA Xfer Error callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_SRAM_WriteOperation_Enable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)

Function description

Enables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_WriteOperation_Disable

Function name

HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)

Function description

Disables dynamically SRAM write operation.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** status

HAL_SRAM_GetState

Function name

HAL_SRAM_StateTypeDef HAL_SRAM_GetState (const SRAM_HandleTypeDef * hsram)

Function description

Returns the SRAM controller state.

Parameters

- **hsram:** pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.

Return values

- **HAL:** state

80.3 SRAM Firmware driver defines

The following section lists the various define and macros of the module.

80.3.1 SRAM

SRAM

SRAM Exported Macros

__HAL_SRAM_RESET_HANDLE_STATE

Description:

- Reset SRAM handle state.

Parameters:

- __HANDLE__: SRAM handle

Return value:

- None

81 HAL TIM Generic Driver

81.1 TIM Firmware driver registers structures

81.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Period***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***
- ***uint32_t AutoReloadPreload***

Field Documentation

- ***uint32_t TIM_Base_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF Macro **__HAL_TIM_CALC_PSC()** can be used to calculate prescaler value
- ***uint32_t TIM_Base_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_Counter_Mode***
- ***uint32_t TIM_Base_InitTypeDef::Period***
  Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated)Macros **__HAL_TIM_CALC_PERIOD()**, **__HAL_TIM_CALC_PERIOD_DITHER()**,**__HAL_TIM_CALC_PERIOD_BY_DELAY()**, **__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY()**can be used to calculate Period value
- ***uint32_t TIM_Base_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_ClockDivision***
- ***uint32_t TIM_Base_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  – the number of PWM periods in edge-aligned mode
  – the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- ***uint32_t TIM_Base_InitTypeDef::AutoReloadPreload***
  Specifies the auto-reload preload. This parameter can be a value of ***TIM_AutoReloadPreload***

81.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCFastMode***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t TIM_OC_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OC_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***
- ***uint32_t TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_Output_Compare_N_Polarity***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
  Specifies the Fast mode state. This parameter can be a value of ***TIM_Output_Fast_State***
  Note:
  – This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_Output_Compare_N_Idle_State***
  Note:
  – This parameter is valid only for timer instances supporting break feature.

81.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
  Specifies the TIM mode. This parameter can be a value of ***TIM_Output_Compare_and_PWM_modes***
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF (or 0xFFEF if dithering is activated) Macros **__HAL_TIM_CALC_PULSE()**, **__HAL_TIM_CALC_PULSE_DITHER()** can be used to calculate Pulse value
- ***uint32_t TIM_OnePulse_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_Output_Compare_Polarity***

- ***uint32_t TIM_OnePulse_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of *TIM_Output_Compare_N_Polarity*
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of *TIM_Output_Compare_Idle_State*
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of *TIM_Output_Compare_N_Idle_State*
  Note:
  – This parameter is valid only for timer instances supporting break feature.
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of *TIM_Input_Capture_Polarity*
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of *TIM_Input_Capture_Selection*
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

81.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of *TIM_Input_Capture_Polarity*
- ***uint32_t TIM_IC_InitTypeDef::ICSelection***
  Specifies the input. This parameter can be a value of *TIM_Input_Capture_Selection*
- ***uint32_t TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of *TIM_Input_Capture_Prescaler*
- ***uint32_t TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

81.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1Selection***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2Selection***
- ***uint32_t IC2Prescaler***

- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t TIM_Encoder_InitTypeDef::EncoderMode***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Mode***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Encoder_Input_Polarity***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
  Specifies the input. This parameter can be a value of ***TIM_Input_Capture_Selection***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

81.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
  TIM clock sources This parameter can be a value of ***TIM_Clock_Source***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
  TIM clock polarity This parameter can be a value of ***TIM_Clock_Polarity***
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
  TIM clock prescaler This parameter can be a value of ***TIM_Clock_Prescaler***
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
  TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

81.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
  TIM clear Input state This parameter can be ENABLE or DISABLE

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
  TIM clear Input sources This parameter can be a value of ***TIM_ClearInput_Source***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
  TIM Clear Input polarity This parameter can be a value of ***TIM_ClearInput_Polarity***
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
  TIM Clear Input prescaler This parameter must be 0: When OCRef clear feature is used with ETR source, ETR prescaler must be off
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
  TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

81.1.8 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t MasterOutputTrigger***
- ***uint32_t MasterOutputTrigger2***
- ***uint32_t MasterSlaveMode***

Field Documentation

- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger***
  Trigger output (TRGO) selection This parameter can be a value of ***TIM_Master_Mode_Selection***
- ***uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger2***
  Trigger output2 (TRGO2) selection This parameter can be a value of ***TIM_Master_Mode_Selection_2***
- ***uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode***
  Master/slave mode selection This parameter can be a value of ***TIM_Master_Slave_Mode***
  Note:
  – When the Master/slave mode is enabled, the effect of an event on the trigger input (TRGI) is delayed to allow a perfect synchronization between the current timer and its slaves (through TRGO). It is not mandatory in case of timer synchronization mode.

81.1.9 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
  Slave mode selection This parameter can be a value of ***TIM_Slave_Mode***
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
  Input Trigger source This parameter can be a value of ***TIM_Trigger_Selection***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
  Input Trigger polarity This parameter can be a value of ***TIM_Trigger_Polarity***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
  Input trigger prescaler This parameter can be a value of ***TIM_Trigger_Prescaler***
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
  Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

81.1.10 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***uint32_t OffStateRunMode***
- ***uint32_t OffStateIDLEMode***

- ***uint32_t LockLevel***
- ***uint32_t DeadTime***
- ***uint32_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateRunMode***
  TIM off state in run mode, This parameter can be a value of ***TIM_OSSR_Off_State_Selection_for_Run_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::OffStateIDLEMode***
  TIM off state in IDLE mode, This parameter can be a value of ***TIM_OSSI_Off_State_Selection_for_Idle_mode_state***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::LockLevel***
  TIM Lock level, This parameter can be a value of ***TIM_Lock_level***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::DeadTime***
  TIM dead Time, This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakState***
  TIM Break State, This parameter can be a value of ***TIM_Break_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakPolarity***
  TIM Break input polarity, This parameter can be a value of ***TIM_Break_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakFilter***
  Specifies the break input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input.This parameter can be a value of ***TIM_Break_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2State***
  TIM Break2 State, This parameter can be a value of ***TIM_Break2_Input_enable_disable***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Polarity***
  TIM Break2 input polarity, This parameter can be a value of ***TIM_Break2_Polarity***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2Filter***
  TIM break2 input filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input.This parameter can be a value of ***TIM_Break2_Input_AF_Mode***
- ***uint32_t TIM_BreakDeadTimeConfigTypeDef::AutomaticOutput***
  TIM Automatic Output Enable state, This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

81.1.11 __TIM_HandleTypeDef

__TIM_HandleTypeDef is defined in the stm32u5xx_hal_tim.h

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***
- ***__IO HAL_TIM_ChannelStateTypeDef ChannelState***

- *__IO HAL_TIM_ChannelStateTypeDef ChannelNState*
- *__IO HAL_TIM_DMABurstStateTypeDef DMABurstState*
- *void(* Base_MspInitCallback*
- *void(* Base_MspDeInitCallback*
- *void(* IC_MspInitCallback*
- *void(* IC_MspDeInitCallback*
- *void(* OC_MspInitCallback*
- *void(* OC_MspDeInitCallback*
- *void(* PWM_MspInitCallback*
- *void(* PWM_MspDeInitCallback*
- *void(* OnePulse_MspInitCallback*
- *void(* OnePulse_MspDeInitCallback*
- *void(* Encoder_MspInitCallback*
- *void(* Encoder_MspDeInitCallback*
- *void(* HallSensor_MspInitCallback*
- *void(* HallSensor_MspDeInitCallback*
- *void(* PeriodElapsedCallback*
- *void(* PeriodElapsedHalfCpltCallback*
- *void(* TriggerCallback*
- *void(* TriggerHalfCpltCallback*
- *void(* IC_CaptureCallback*
- *void(* IC_CaptureHalfCpltCallback*
- *void(* OC_DelayElapsedCallback*
- *void(* PWM_PulseFinishedCallback*
- *void(* PWM_PulseFinishedHalfCpltCallback*
- *void(* ErrorCallback*
- *void(* CommutationCallback*
- *void(* CommutationHalfCpltCallback*
- *void(* BreakCallback*
- *void(* Break2Callback*
- *void(* EncoderIndexCallback*
- *void(* DirectionChangeCallback*
- *void(* IndexErrorCallback*
- *void(* TransitionErrorCallback*

Field Documentation

- ***TIM_TypeDef* __TIM_HandleTypeDef::Instance***
  Register base address
- ***TIM_Base_InitTypeDef __TIM_HandleTypeDef::Init***
  TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel __TIM_HandleTypeDef::Channel***
  Active channel
- ***DMA_HandleTypeDef* __TIM_HandleTypeDef::hdma[7]***
  DMA Handlers array This array is accessed by a ***DMA_Handle_index***
- ***HAL_LockTypeDef __TIM_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_TIM_StateTypeDef __TIM_HandleTypeDef::State***
  TIM operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelState[6]***
  TIM channel operation state
- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelNState[4]***
  TIM complementary channel operation state

- ***__IO HAL_TIM_DMABurstStateTypeDef __TIM_HandleTypeDef::DMABurstState***
  DMA burst operation state
- ***void(* __TIM_HandleTypeDef::Base_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Base_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Base Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::IC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::IC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM IC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OC_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OC_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM OC Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp Init Callback
- ***void(* __TIM_HandleTypeDef::PWM_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp Init Callback
- ***void(* __TIM_HandleTypeDef::OnePulse_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM One Pulse Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp Init Callback
- ***void(* __TIM_HandleTypeDef::Encoder_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp Init Callback
- ***void(* __TIM_HandleTypeDef::HallSensor_MspDeInitCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Hall Sensor Msp DeInit Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PeriodElapsedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Period Elapsed half complete Callback
- ***void(* __TIM_HandleTypeDef::TriggerCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger Callback
- ***void(* __TIM_HandleTypeDef::TriggerHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Trigger half complete Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture Callback
- ***void(* __TIM_HandleTypeDef::IC_CaptureHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Input Capture half complete Callback
- ***void(* __TIM_HandleTypeDef::OC_DelayElapsedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Output Compare Delay Elapsed Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished Callback
- ***void(* __TIM_HandleTypeDef::PWM_PulseFinishedHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM PWM Pulse Finished half complete Callback
- ***void(* __TIM_HandleTypeDef::ErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Error Callback
- ***void(* __TIM_HandleTypeDef::CommutationCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation Callback
- ***void(* __TIM_HandleTypeDef::CommutationHalfCpltCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Commutation half complete Callback

- ***void(* __TIM_HandleTypeDef::BreakCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break Callback
- ***void(* __TIM_HandleTypeDef::Break2Callback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break2 Callback
- ***void(* __TIM_HandleTypeDef::EncoderIndexCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Encoder Index Callback
- ***void(* __TIM_HandleTypeDef::DirectionChangeCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Direction Change Callback
- ***void(* __TIM_HandleTypeDef::IndexErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Index Error Callback
- ***void(* __TIM_HandleTypeDef::TransitionErrorCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Transition Error Callback

81.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

81.2.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
   – Input Capture
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental encoder for positioning purposes

81.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Time Base : HAL_TIM_Base_MspInit()
   – Input Capture : HAL_TIM_IC_MspInit()
   – Output Compare : HAL_TIM_OC_MspInit()
   – PWM generation : HAL_TIM_PWM_MspInit()
   – One-pulse mode output : HAL_TIM_OnePulse_MspInit()
   – Encoder mode output : HAL_TIM_Encoder_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.

4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
   – HAL_TIM_Base_Init: to use the Timer to generate a simple time base
   – HAL_TIM_OC_Init, HAL_TIM_OC_ConfigChannel and optionally HAL_TIMEx_OC_ConfigPulseOnCompare: to use the Timer to generate an Output Compare signal.
   – HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
   – HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
   – HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
   – HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
   – Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
   – Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
   – Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
   – PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
   – One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
   – Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions: HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

Callback registration

The compilation define USE_HAL_TIM_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_TIM_RegisterCallback() to register a callback. HAL_TIM_RegisterCallback() takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_TIM_UnRegisterCallback() to reset a callback to the default weak function. HAL_TIM_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

These functions allow to register/unregister following callbacks:

- Base_MspInitCallback : TIM Base Msp Init Callback.
- Base_MspDeInitCallback : TIM Base Msp DeInit Callback.
- IC_MspInitCallback : TIM IC Msp Init Callback.
- IC_MspDeInitCallback : TIM IC Msp DeInit Callback.
- OC_MspInitCallback : TIM OC Msp Init Callback.
- OC_MspDeInitCallback : TIM OC Msp DeInit Callback.
- PWM_MspInitCallback : TIM PWM Msp Init Callback.
- PWM_MspDeInitCallback : TIM PWM Msp DeInit Callback.
- OnePulse_MspInitCallback : TIM One Pulse Msp Init Callback.
- OnePulse_MspDeInitCallback : TIM One Pulse Msp DeInit Callback.
- Encoder_MspInitCallback : TIM Encoder Msp Init Callback.
- Encoder_MspDeInitCallback : TIM Encoder Msp DeInit Callback.
- HallSensor_MspInitCallback : TIM Hall Sensor Msp Init Callback.
- HallSensor_MspDeInitCallback : TIM Hall Sensor Msp DeInit Callback.
- PeriodElapsedCallback : TIM Period Elapsed Callback.
- PeriodElapsedHalfCpltCallback : TIM Period Elapsed half complete Callback.
- TriggerCallback : TIM Trigger Callback.
- TriggerHalfCpltCallback : TIM Trigger half complete Callback.
- IC_CaptureCallback : TIM Input Capture Callback.
- IC_CaptureHalfCpltCallback : TIM Input Capture half complete Callback.
- OC_DelayElapsedCallback : TIM Output Compare Delay Elapsed Callback.
- PWM_PulseFinishedCallback : TIM PWM Pulse Finished Callback.
- PWM_PulseFinishedHalfCpltCallback : TIM PWM Pulse Finished half complete Callback.

- ErrorCallback : TIM Error Callback.
- CommutationCallback : TIM Commutation Callback.
- CommutationHalfCpltCallback : TIM Commutation half complete Callback.
- BreakCallback : TIM Break Callback.
- Break2Callback : TIM Break2 Callback.
- EncoderIndexCallback : TIM Encoder Index Callback.
- DirectionChangeCallback : TIM Direction Change Callback
- IndexErrorCallback : TIM Index Error Callback.
- TransitionErrorCallback : TIM Transition Error Callback

By default, after the Init and when the state is HAL_TIM_STATE_RESET all interrupt callbacks are set to the corresponding weak functions: examples HAL_TIM_TriggerCallback(), HAL_TIM_ErrorCallback().

Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functionalities in the Init / DeInit only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the Init / DeInit keep and use the user MspInit / MspDeInit callbacks(registered beforehand)

Callbacks can be registered / unregistered in HAL_TIM_STATE_READY state only. Exception done MspInit / MspDeInit that can be registered / unregistered in HAL_TIM_STATE_READY or HAL_TIM_STATE_RESET state, thus registered(user) MspInit / DeInit callbacks can be used during the Init / DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_TIM_RegisterCallback() before calling DeInit or Init function.

When The compilation define USE_HAL_TIM_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

81.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- ***HAL_TIM_Base_Init()***
- ***HAL_TIM_Base_DeInit()***
- ***HAL_TIM_Base_MspInit()***
- ***HAL_TIM_Base_MspDeInit()***
- ***HAL_TIM_Base_Start()***
- ***HAL_TIM_Base_Stop()***
- ***HAL_TIM_Base_Start_IT()***
- ***HAL_TIM_Base_Stop_IT()***
- ***HAL_TIM_Base_Start_DMA()***
- ***HAL_TIM_Base_Stop_DMA()***

81.2.4 TIM Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.
- De-initialize the TIM Output Compare.
- Start the TIM Output Compare.
- Stop the TIM Output Compare.
- Start the TIM Output Compare and enable interrupt.
- Stop the TIM Output Compare and disable interrupt.

- Start the TIM Output Compare and enable DMA transfer.
- Stop the TIM Output Compare and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_OC_Init()*
- *HAL_TIM_OC_DeInit()*
- *HAL_TIM_OC_MspInit()*
- *HAL_TIM_OC_MspDeInit()*
- *HAL_TIM_OC_Start()*
- *HAL_TIM_OC_Stop()*
- *HAL_TIM_OC_Start_IT()*
- *HAL_TIM_OC_Stop_IT()*
- *HAL_TIM_OC_Start_DMA()*
- *HAL_TIM_OC_Stop_DMA()*

81.2.5 TIM PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the TIM PWM.
- Stop the TIM PWM.
- Start the TIM PWM and enable interrupt.
- Stop the TIM PWM and disable interrupt.
- Start the TIM PWM and enable DMA transfer.
- Stop the TIM PWM and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_PWM_Init()*
- *HAL_TIM_PWM_DeInit()*
- *HAL_TIM_PWM_MspInit()*
- *HAL_TIM_PWM_MspDeInit()*
- *HAL_TIM_PWM_Start()*
- *HAL_TIM_PWM_Stop()*
- *HAL_TIM_PWM_Start_IT()*
- *HAL_TIM_PWM_Stop_IT()*
- *HAL_TIM_PWM_Start_DMA()*
- *HAL_TIM_PWM_Stop_DMA()*

81.2.6 TIM Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the TIM Input Capture.
- Stop the TIM Input Capture.
- Start the TIM Input Capture and enable interrupt.
- Stop the TIM Input Capture and disable interrupt.
- Start the TIM Input Capture and enable DMA transfer.
- Stop the TIM Input Capture and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_IC_Init()*
- *HAL_TIM_IC_DeInit()*

- *HAL_TIM_IC_MspInit()*
- *HAL_TIM_IC_MspDeInit()*
- *HAL_TIM_IC_Start()*
- *HAL_TIM_IC_Stop()*
- *HAL_TIM_IC_Start_IT()*
- *HAL_TIM_IC_Stop_IT()*
- *HAL_TIM_IC_Start_DMA()*
- *HAL_TIM_IC_Stop_DMA()*

81.2.7 TIM One Pulse functions

This section provides functions allowing to:
- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the TIM One Pulse.
- Stop the TIM One Pulse.
- Start the TIM One Pulse and enable interrupt.
- Stop the TIM One Pulse and disable interrupt.
- Start the TIM One Pulse and enable DMA transfer.
- Stop the TIM One Pulse and disable DMA transfer.

This section contains the following APIs:
- *HAL_TIM_OnePulse_Init()*
- *HAL_TIM_OnePulse_DeInit()*
- *HAL_TIM_OnePulse_MspInit()*
- *HAL_TIM_OnePulse_MspDeInit()*
- *HAL_TIM_OnePulse_Start()*
- *HAL_TIM_OnePulse_Stop()*
- *HAL_TIM_OnePulse_Start_IT()*
- *HAL_TIM_OnePulse_Stop_IT()*

81.2.8 TIM Encoder functions

This section provides functions allowing to:
- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the TIM Encoder.
- Stop the TIM Encoder.
- Start the TIM Encoder and enable interrupt.
- Stop the TIM Encoder and disable interrupt.
- Start the TIM Encoder and enable DMA transfer.
- Stop the TIM Encoder and disable DMA transfer.

This section contains the following APIs:
- *HAL_TIM_Encoder_Init()*
- *HAL_TIM_Encoder_DeInit()*
- *HAL_TIM_Encoder_MspInit()*
- *HAL_TIM_Encoder_MspDeInit()*
- *HAL_TIM_Encoder_Start()*
- *HAL_TIM_Encoder_Stop()*
- *HAL_TIM_Encoder_Start_IT()*
- *HAL_TIM_Encoder_Stop_IT()*
- *HAL_TIM_Encoder_Start_DMA()*

- *HAL_TIM_Encoder_Stop_DMA()*

81.2.9 TIM Callbacks functions

This section provides TIM callback functions:

- TIM Period elapsed callback
- TIM Output Compare callback
- TIM Input capture callback
- TIM Trigger callback
- TIM Error callback
- TIM Index callback
- TIM Direction change callback
- TIM Index error callback
- TIM Transition error callback

This section contains the following APIs:

- *HAL_TIM_PeriodElapsedCallback()*
- *HAL_TIM_PeriodElapsedHalfCpltCallback()*
- *HAL_TIM_OC_DelayElapsedCallback()*
- *HAL_TIM_IC_CaptureCallback()*
- *HAL_TIM_IC_CaptureHalfCpltCallback()*
- *HAL_TIM_PWM_PulseFinishedCallback()*
- *HAL_TIM_PWM_PulseFinishedHalfCpltCallback()*
- *HAL_TIM_TriggerCallback()*
- *HAL_TIM_TriggerHalfCpltCallback()*
- *HAL_TIM_ErrorCallback()*
- *HAL_TIM_RegisterCallback()*
- *HAL_TIM_UnRegisterCallback()*

81.2.10 Detailed description of functions

HAL_TIM_Base_Init

Function name

HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and initialize the associated handle.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Base_DeInit() before HAL_TIM_Base_Init()

HAL_TIM_Base_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Base peripheral.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_MspInit

Function name

void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_MspDeInit

Function name

void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Base MSP.

Parameters

- **htim:** TIM Base handle

Return values

- **None:**

HAL_TIM_Base_Start

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in interrupt mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_Base_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Start_DMA (TIM_HandleTypeDef * htim, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to peripheral.

Return values

- **HAL:** status

HAL_TIM_Base_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Base generation in DMA mode.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** status

HAL_TIM_OC_Init

Function name

HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OC_DeInit() before HAL_TIM_OC_Init()

HAL_TIM_OC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** status

HAL_TIM_OC_MspInit

Function name

void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_MspDeInit

Function name

void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Output Compare MSP.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **None:**

HAL_TIM_OC_Start

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
  – TIM_CHANNEL_5: TIM Channel 5 selected
  – TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
  – TIM_CHANNEL_5: TIM Channel 5 selected
  – TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_OC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Init

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_PWM_DeInit() before HAL_TIM_PWM_Init()

HAL_TIM_PWM_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM PWM handle

Return values

- **HAL:** status

HAL_TIM_PWM_MspInit

Function name

void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_MspDeInit

Function name

void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM PWM MSP.

Parameters

- **htim:** TIM PWM handle

Return values

- **None:**

HAL_TIM_PWM_Start

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
  – TIM_CHANNEL_5: TIM Channel 5 selected
  – TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_PWM_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIM_PWM_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** TIM PWM handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Init

Function name

HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_IC_DeInit() before HAL_TIM_IC_Init()

HAL_TIM_IC_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM peripheral.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **HAL:** status

HAL_TIM_IC_MspInit

Function name

void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Input Capture MSP.

Parameters

- **htim:** TIM Input Capture handle

Return values

- **None:**

HAL_TIM_IC_MspDeInit

Function name

void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Input Capture MSP.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_IC_Start

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in interrupt mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_IC_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_IC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** TIM Input Capture handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_Init

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)

Function description

Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and initializes the associated handle.

Parameters

- **htim:** TIM One Pulse handle
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values:
  - TIM_OPMODE_SINGLE: Only one pulse will be generated.
  - TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_OnePulse_DeInit() before HAL_TIM_OnePulse_Init()
- When the timer instance is initialized in One Pulse mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_OnePulse_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM One Pulse.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **HAL:** status

HAL_TIM_OnePulse_MspInit

Function name

void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_MspDeInit

Function name

void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM One Pulse MSP.

Parameters

- **htim:** TIM One Pulse handle

Return values

- **None:**

HAL_TIM_OnePulse_Start

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Stop

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_OnePulse_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** See note above

Return values

- **HAL:** status

Notes

- Though OutputChannel parameter is deprecated and ignored by the function it has been kept to avoid HAL_TIM API compatibility break.
- The pulse output channel is determined when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIM_Encoder_Init

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, const TIM_Encoder_InitTypeDef * sConfig)

Function description

Initializes the TIM Encoder Interface and initialize the associated handle.

Parameters

- **htim:** TIM Encoder Interface handle
- **sConfig:** TIM Encoder Interface configuration structure

Return values

- **HAL:** status

Notes

- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode. Ex: call HAL_TIM_Encoder_DeInit() before HAL_TIM_Encoder_Init()
- Encoder mode and External clock mode 2 are not compatible and must not be selected together Ex: A call for HAL_TIM_Encoder_Init will erase the settings of HAL_TIM_ConfigClockSource using TIM_CLOCKSOURCE_ETRMODE2 and vice versa
- When the timer instance is initialized in Encoder mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIM_Encoder_DeInit

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Encoder interface.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** status

HAL_TIM_Encoder_MspInit

Function name

void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_MspDeInit

Function name

void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Encoder Interface MSP.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **None:**

HAL_TIM_Encoder_Start

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in interrupt mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_Encoder_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)

Function description

Starts the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
- **pData1:** The destination Buffer address for IC1.
- **pData2:** The destination Buffer address for IC2.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIM_Encoder_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Encoder Interface in DMA mode.

Parameters

- **htim:** TIM Encoder Interface handle
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected

Return values

- **HAL:** status

HAL_TIM_IRQHandler

Function name

void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

Function description

This function handles TIM interrupts requests.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM Output Compare handle
- **sConfig:** TIM Output Compare configuration structure
- **Channel:** TIM Channels to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_PWM_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim:** TIM PWM handle
- **sConfig:** TIM PWM configuration structure
- **Channel:** TIM Channels to be configured This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
  – TIM_CHANNEL_5: TIM Channel 5 selected
  – TIM_CHANNEL_6: TIM Channel 6 selected

Return values

- **HAL:** status

HAL_TIM_IC_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel (TIM_HandleTypeDef * htim, const TIM_IC_InitTypeDef * sConfig, uint32_t Channel)

Function description

Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters

- **htim:** TIM IC handle
- **sConfig:** TIM Input Capture configuration structure
- **Channel:** TIM Channel to configure This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIM_OnePulse_ConfigChannel

Function name

HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)

Function description

Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim:** TIM One Pulse handle
- **sConfig:** TIM One Pulse configuration structure
- **OutputChannel:** TIM output channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
- **InputChannel:** TIM input Channel to configure This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- To output a waveform with a minimum delay user can enable the fast mode by calling the __HAL_TIM_ENABLE_OCxFAST macro. Then CCx output is forced in response to the edge detection on TIx input, without taking in account the comparison.

HAL_TIM_ConfigOCrefClear

Function name

HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, const TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)

Function description

Configures the OCRef clear feature.

Parameters

- **htim:** TIM handle
- **sClearInputConfig:** pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREF clear feature and parameters for the TIM peripheral.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **HAL:** status

HAL_TIM_ConfigClockSource

Function name

HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, const TIM_ClockConfigTypeDef * sClockSourceConfig)

Function description

Configures the clock source to be used.

Parameters

- **htim:** TIM handle
- **sClockSourceConfig:** pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.

Return values

- **HAL:** status

HAL_TIM_ConfigTI1Input

Function name

HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)

Function description

Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.

Parameters

- **htim:** TIM handle.
- **TI1_Selection:** Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values:
  – TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 input
  – TIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1, Reset + Trigger, Gated + Reset).

Return values

- **HAL:** status

HAL_TIM_SlaveConfigSynchro_IT

Function name

HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchro_IT (TIM_HandleTypeDef * htim, const TIM_SlaveConfigTypeDef * sSlaveConfig)

Function description

Configures the TIM in Slave mode in interrupt mode.

Parameters

- **htim:** TIM handle.
- **sSlaveConfig:** pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1, Reset + Trigger, Gated + Reset).

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  – TIM_DMABASE_CR1
  – TIM_DMABASE_CR2
  – TIM_DMABASE_SMCR
  – TIM_DMABASE_DIER
  – TIM_DMABASE_SR
  – TIM_DMABASE_EGR
  – TIM_DMABASE_CCMR1
  – TIM_DMABASE_CCMR2
  – TIM_DMABASE_CCER
  – TIM_DMABASE_CNT
  – TIM_DMABASE_PSC
  – TIM_DMABASE_ARR
  – TIM_DMABASE_RCR
  – TIM_DMABASE_CCR1
  – TIM_DMABASE_CCR2
  – TIM_DMABASE_CCR3
  – TIM_DMABASE_CCR4
  – TIM_DMABASE_BDTR
  – TIM_DMABASE_CCMR3
  – TIM_DMABASE_CCR5
  – TIM_DMABASE_CCR6
  – TIM_DMABASE_DTR2
  – TIM_DMABASE_ECR
  – TIM_DMABASE_TISEL
  – TIM_DMABASE_AF1
  – TIM_DMABASE_AF2
  – TIM_DMABASE_OR1
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  – TIM_DMA_UPDATE: TIM update Interrupt source
  – TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  – TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  – TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  – TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  – TIM_DMA_COM: TIM Commutation DMA source
  – TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiWriteStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiWriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, const uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer multiple Data from the memory to the TIM peripheral.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
  – TIM_DMABASE_CR1
  – TIM_DMABASE_CR2
  – TIM_DMABASE_SMCR
  – TIM_DMABASE_DIER
  – TIM_DMABASE_SR
  – TIM_DMABASE_EGR
  – TIM_DMABASE_CCMR1
  – TIM_DMABASE_CCMR2
  – TIM_DMABASE_CCER
  – TIM_DMABASE_CNT
  – TIM_DMABASE_PSC
  – TIM_DMABASE_ARR
  – TIM_DMABASE_RCR
  – TIM_DMABASE_CCR1
  – TIM_DMABASE_CCR2
  – TIM_DMABASE_CCR3
  – TIM_DMABASE_CCR4
  – TIM_DMABASE_BDTR
  – TIM_DMABASE_CCMR3
  – TIM_DMABASE_CCR5
  – TIM_DMABASE_CCR6
  – TIM_DMABASE_DTR2
  – TIM_DMABASE_ECR
  – TIM_DMABASE_TISEL
  – TIM_DMABASE_AF1
  – TIM_DMABASE_AF2
  – TIM_DMABASE_OR1
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  – TIM_DMA_UPDATE: TIM update Interrupt source
  – TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  – TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  – TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  – TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  – TIM_DMA_COM: TIM Commutation DMA source
  – TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_WriteStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stops the TIM DMA Burst mode.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  – TIM_DMABASE_CR1
  – TIM_DMABASE_CR2
  – TIM_DMABASE_SMCR
  – TIM_DMABASE_DIER
  – TIM_DMABASE_SR
  – TIM_DMABASE_EGR
  – TIM_DMABASE_CCMR1
  – TIM_DMABASE_CCMR2
  – TIM_DMABASE_CCER
  – TIM_DMABASE_CNT
  – TIM_DMABASE_PSC
  – TIM_DMABASE_ARR
  – TIM_DMABASE_RCR
  – TIM_DMABASE_CCR1
  – TIM_DMABASE_CCR2
  – TIM_DMABASE_CCR3
  – TIM_DMABASE_CCR4
  – TIM_DMABASE_BDTR
  – TIM_DMABASE_CCMR3
  – TIM_DMABASE_CCR5
  – TIM_DMABASE_CCR6
  – TIM_DMABASE_DTR2
  – TIM_DMABASE_ECR
  – TIM_DMABASE_TISEL
  – TIM_DMABASE_AF1
  – TIM_DMABASE_AF2
  – TIM_DMABASE_OR1
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  – TIM_DMA_UPDATE: TIM update Interrupt source
  – TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  – TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  – TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  – TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  – TIM_DMA_COM: TIM Commutation DMA source
  – TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.

Return values

- **HAL:** status

Notes

- This function should be used only when BurstLength is equal to DMA data transfer length.

HAL_TIM_DMABurst_MultiReadStart

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_MultiReadStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength, uint32_t DataLength)

Function description

Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

Parameters

- **htim:** TIM handle
- **BurstBaseAddress:** TIM Base address from where the DMA will start the Data read This parameter can be one of the following values:
  - TIM_DMABASE_CR1
  - TIM_DMABASE_CR2
  - TIM_DMABASE_SMCR
  - TIM_DMABASE_DIER
  - TIM_DMABASE_SR
  - TIM_DMABASE_EGR
  - TIM_DMABASE_CCMR1
  - TIM_DMABASE_CCMR2
  - TIM_DMABASE_CCER
  - TIM_DMABASE_CNT
  - TIM_DMABASE_PSC
  - TIM_DMABASE_ARR
  - TIM_DMABASE_RCR
  - TIM_DMABASE_CCR1
  - TIM_DMABASE_CCR2
  - TIM_DMABASE_CCR3
  - TIM_DMABASE_CCR4
  - TIM_DMABASE_BDTR
  - TIM_DMABASE_CCMR3
  - TIM_DMABASE_CCR5
  - TIM_DMABASE_CCR6
  - TIM_DMABASE_DTR2
  - TIM_DMABASE_ECR
  - TIM_DMABASE_TISEL
  - TIM_DMABASE_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_OR1
- **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values:
  - TIM_DMA_UPDATE: TIM update Interrupt source
  - TIM_DMA_CC1: TIM Capture Compare 1 DMA source
  - TIM_DMA_CC2: TIM Capture Compare 2 DMA source
  - TIM_DMA_CC3: TIM Capture Compare 3 DMA source
  - TIM_DMA_CC4: TIM Capture Compare 4 DMA source
  - TIM_DMA_COM: TIM Commutation DMA source
  - TIM_DMA_TRIGGER: TIM Trigger DMA source
- **BurstBuffer:** The Buffer address.
- **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_26TRANSFER.
- **DataLength:** Data length. This parameter can be one value between 1 and 0xFFFF.

Return values

- **HAL:** status

HAL_TIM_DMABurst_ReadStop

Function name

HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)

Function description

Stop the DMA burst reading.

Parameters

- **htim:** TIM handle
- **BurstRequestSrc:** TIM DMA Request sources to disable.

Return values

- **HAL:** status

HAL_TIM_GenerateEvent

Function name

HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)

Function description

Generate a software event.

Parameters

- **htim:** TIM handle
- **EventSource:** specifies the event source. This parameter can be one of the following values:
  – TIM_EVENTSOURCE_UPDATE: Timer update Event source
  – TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source
  – TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source
  – TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source
  – TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source
  – TIM_EVENTSOURCE_COM: Timer COM event source
  – TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
  – TIM_EVENTSOURCE_BREAK: Timer Break event source
  – TIM_EVENTSOURCE_BREAK2: Timer Break2 event source

Return values

- **HAL:** status

Notes

- Basic timers can only generate an update event.
- TIM_EVENTSOURCE_COM is relevant only with advanced timer instances.
- TIM_EVENTSOURCE_BREAK and TIM_EVENTSOURCE_BREAK2 are relevant only for timer instances supporting break input(s).

HAL_TIM_ReadCapturedValue

Function name

uint32_t HAL_TIM_ReadCapturedValue (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Read the captured value from Capture Compare unit.

Parameters

- **htim:** TIM handle.
- **Channel:** TIM Channels to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **Captured:** value

HAL_TIM_PeriodElapsedCallback

Function name

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PeriodElapsedHalfCpltCallback

Function name

void HAL_TIM_PeriodElapsedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Period elapsed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_OC_DelayElapsedCallback

Function name

void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)

Function description

Output Compare callback in non-blocking mode.

Parameters

- **htim:** TIM OC handle

Return values

- **None:**

HAL_TIM_IC_CaptureCallback

Function name

void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_IC_CaptureHalfCpltCallback

Function name

void HAL_TIM_IC_CaptureHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Input Capture half complete callback in non-blocking mode.

Parameters

- **htim:** TIM IC handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedCallback

Function name

void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_PWM_PulseFinishedHalfCpltCallback

Function name

void HAL_TIM_PWM_PulseFinishedHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

PWM Pulse finished half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerCallback

Function name

void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_TriggerHalfCpltCallback

Function name

void HAL_TIM_TriggerHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall Trigger detection half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_ErrorCallback

Function name

void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

Function description

Timer error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIM_RegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_RegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID, pTIM_CallbackTypeDef pCallback)

Function description

Register a User TIM callback to be used instead of the weak predefined callback.

Parameters

- **htim:** tim handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_TIM_BASE_MSPINIT_CB_ID Base MspInit Callback ID
  - HAL_TIM_BASE_MSPDEINIT_CB_ID Base MspDeInit Callback ID
  - HAL_TIM_IC_MSPINIT_CB_ID IC MspInit Callback ID
  - HAL_TIM_IC_MSPDEINIT_CB_ID IC MspDeInit Callback ID
  - HAL_TIM_OC_MSPINIT_CB_ID OC MspInit Callback ID
  - HAL_TIM_OC_MSPDEINIT_CB_ID OC MspDeInit Callback ID
  - HAL_TIM_PWM_MSPINIT_CB_ID PWM MspInit Callback ID
  - HAL_TIM_PWM_MSPDEINIT_CB_ID PWM MspDeInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPINIT_CB_ID One Pulse MspInit Callback ID
  - HAL_TIM_ONE_PULSE_MSPDEINIT_CB_ID One Pulse MspDeInit Callback ID
  - HAL_TIM_ENCODER_MSPINIT_CB_ID Encoder MspInit Callback ID
  - HAL_TIM_ENCODER_MSPDEINIT_CB_ID Encoder MspDeInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPINIT_CB_ID Hall Sensor MspInit Callback ID
  - HAL_TIM_HALL_SENSOR_MSPDEINIT_CB_ID Hall Sensor MspDeInit Callback ID
  - HAL_TIM_PERIOD_ELAPSED_CB_ID Period Elapsed Callback ID
  - HAL_TIM_PERIOD_ELAPSED_HALF_CB_ID Period Elapsed half complete Callback ID
  - HAL_TIM_TRIGGER_CB_ID Trigger Callback ID
  - HAL_TIM_TRIGGER_HALF_CB_ID Trigger half complete Callback ID
  - HAL_TIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  - HAL_TIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  - HAL_TIM_OC_DELAY_ELAPSED_CB_ID Output Compare Delay Elapsed Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_CB_ID PWM Pulse Finished Callback ID
  - HAL_TIM_PWM_PULSE_FINISHED_HALF_CB_ID PWM Pulse Finished half complete Callback ID
  - HAL_TIM_ERROR_CB_ID Error Callback ID
  - HAL_TIM_COMMUTATION_CB_ID Commutation Callback ID
  - HAL_TIM_COMMUTATION_HALF_CB_ID Commutation half complete Callback ID
  - HAL_TIM_BREAK_CB_ID Break Callback ID
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID
  - HAL_TIM_ENCODER_INDEX_CB_ID Encoder Index Callback ID
  - HAL_TIM_DIRECTION_CHANGE_CB_ID Direction Change Callback ID
  - HAL_TIM_INDEX_ERROR_CB_ID Index Error Callback ID
  - HAL_TIM_TRANSITION_ERROR_CB_ID Transition Error Callback ID
- **pCallback:** pointer to the callback function

Return values

- **status:**

HAL_TIM_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_TIM_UnRegisterCallback (TIM_HandleTypeDef * htim, HAL_TIM_CallbackIDTypeDef CallbackID)

Function description

Unregister a TIM callback TIM callback is redirected to the weak predefined callback.

Parameters

- **htim:** tim handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_TIM_BASE_MSPINIT_CB_ID Base MspInit Callback ID
  – HAL_TIM_BASE_MSPDEINIT_CB_ID Base MspDeInit Callback ID
  – HAL_TIM_IC_MSPINIT_CB_ID IC MspInit Callback ID
  – HAL_TIM_IC_MSPDEINIT_CB_ID IC MspDeInit Callback ID
  – HAL_TIM_OC_MSPINIT_CB_ID OC MspInit Callback ID
  – HAL_TIM_OC_MSPDEINIT_CB_ID OC MspDeInit Callback ID
  – HAL_TIM_PWM_MSPINIT_CB_ID PWM MspInit Callback ID
  – HAL_TIM_PWM_MSPDEINIT_CB_ID PWM MspDeInit Callback ID
  – HAL_TIM_ONE_PULSE_MSPINIT_CB_ID One Pulse MspInit Callback ID
  – HAL_TIM_ONE_PULSE_MSPDEINIT_CB_ID One Pulse MspDeInit Callback ID
  – HAL_TIM_ENCODER_MSPINIT_CB_ID Encoder MspInit Callback ID
  – HAL_TIM_ENCODER_MSPDEINIT_CB_ID Encoder MspDeInit Callback ID
  – HAL_TIM_HALL_SENSOR_MSPINIT_CB_ID Hall Sensor MspInit Callback ID
  – HAL_TIM_HALL_SENSOR_MSPDEINIT_CB_ID Hall Sensor MspDeInit Callback ID
  – HAL_TIM_PERIOD_ELAPSED_CB_ID Period Elapsed Callback ID
  – HAL_TIM_PERIOD_ELAPSED_HALF_CB_ID Period Elapsed half complete Callback ID
  – HAL_TIM_TRIGGER_CB_ID Trigger Callback ID
  – HAL_TIM_TRIGGER_HALF_CB_ID Trigger half complete Callback ID
  – HAL_TIM_IC_CAPTURE_CB_ID Input Capture Callback ID
  – HAL_TIM_IC_CAPTURE_HALF_CB_ID Input Capture half complete Callback ID
  – HAL_TIM_OC_DELAY_ELAPSED_CB_ID Output Compare Delay Elapsed Callback ID
  – HAL_TIM_PWM_PULSE_FINISHED_CB_ID PWM Pulse Finished Callback ID
  – HAL_TIM_PWM_PULSE_FINISHED_HALF_CB_ID PWM Pulse Finished half complete Callback ID
  – HAL_TIM_ERROR_CB_ID Error Callback ID
  – HAL_TIM_COMMUTATION_CB_ID Commutation Callback ID
  – HAL_TIM_COMMUTATION_HALF_CB_ID Commutation half complete Callback ID
  – HAL_TIM_BREAK_CB_ID Break Callback ID
  – HAL_TIM_BREAK2_CB_ID Break2 Callback ID
  – HAL_TIM_ENCODER_INDEX_CB_ID Encoder Index Callback ID
  – HAL_TIM_DIRECTION_CHANGE_CB_ID Direction Change Callback ID
  – HAL_TIM_INDEX_ERROR_CB_ID Index Error Callback ID
  – HAL_TIM_TRANSITION_ERROR_CB_ID Transition Error Callback ID

Return values

- **status:**

HAL_TIM_Base_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Base handle state.

Parameters

- **htim:** TIM Base handle

Return values

- **HAL:** state

HAL_TIM_OC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM OC handle state.

Parameters

- **htim:** TIM Output Compare handle

Return values

- **HAL:** state

HAL_TIM_PWM_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM PWM handle state.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** state

HAL_TIM_IC_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Input Capture handle state.

Parameters

- **htim:** TIM IC handle

Return values

- **HAL:** state

HAL_TIM_OnePulse_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM One Pulse Mode handle state.

Parameters

- **htim:** TIM OPM handle

Return values

- **HAL:** state

HAL_TIM_Encoder_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM Encoder Interface handle

Return values

- **HAL:** state

HAL_TIM_GetActiveChannel

Function name

HAL_TIM_ActiveChannel HAL_TIM_GetActiveChannel (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Encoder Mode handle state.

Parameters

- **htim:** TIM handle

Return values

- **Active:** channel

HAL_TIM_GetChannelState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIM_GetChannelState (const TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Return actual state of the TIM channel.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5
  - TIM_CHANNEL_6: TIM Channel 6

Return values

- **TIM:** Channel state

HAL_TIM_DMABurstState

Function name

HAL_TIM_DMABurstStateTypeDef HAL_TIM_DMABurstState (const TIM_HandleTypeDef * htim)

Function description

Return actual state of a DMA burst operation.

Parameters

- **htim:** TIM handle

Return values

- **DMA:** burst state

TIM_Base_SetConfig

Function name

void TIM_Base_SetConfig (TIM_TypeDef * TIMx, const TIM_Base_InitTypeDef * Structure)

Function description

Time Base configuration.

Parameters

- **TIMx:** TIM peripheral
- **Structure:** TIM Base configuration structure

Return values

- **None:**

TIM_TI1_SetConfig

Function name

void TIM_TI1_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ICPolarity, uint32_t TIM_ICSelection, uint32_t TIM_ICFilter)

Function description

Configure the TI1 as Input.

Parameters

- **TIMx:** to select the TIM peripheral.
- **TIM_ICPolarity:** The Input Polarity. This parameter can be one of the following values:
  – TIM_ICPOLARITY_RISING
  – TIM_ICPOLARITY_FALLING
  – TIM_ICPOLARITY_BOTHEDGE
- **TIM_ICSelection:** specifies the input to be used. This parameter can be one of the following values:
  – TIM_ICSELECTION_DIRECTTI: TIM Input 1 is selected to be connected to IC1.
  – TIM_ICSELECTION_INDIRECTTI: TIM Input 1 is selected to be connected to IC2.
  – TIM_ICSELECTION_TRC: TIM Input 1 is selected to be connected to TRC.
- **TIM_ICFilter:** Specifies the Input Capture Filter. This parameter must be a value between 0x00 and 0x0F.

Return values

- **None:**

Notes

- TIM_ICFilter and TIM_ICPolarity are not used in INDIRECT mode as TI2FP1 (on channel2 path) is used as the input signal. Therefore CCMR1 must be protected against un-initialized filter and polarity values.

TIM_OC2_SetConfig

Function name

void TIM_OC2_SetConfig (TIM_TypeDef * TIMx, const TIM_OC_InitTypeDef * OC_Config)

Function description

Timer Output Compare 2 configuration.

Parameters

- **TIMx:** to select the TIM peripheral
- **OC_Config:** The output configuration structure

Return values

- **None:**

TIM_ETR_SetConfig

Function name

void TIM_ETR_SetConfig (TIM_TypeDef * TIMx, uint32_t TIM_ExtTRGPrescaler, uint32_t TIM_ExtTRGPolarity, uint32_t ExtTRGFilter)

Function description

Configures the TIMx External Trigger (ETR).

Parameters

- **TIMx:** to select the TIM peripheral
- **TIM_ExtTRGPrescaler:** The external Trigger Prescaler. This parameter can be one of the following values:
  - TIM_ETRPRESCALER_DIV1: ETRP Prescaler OFF.
  - TIM_ETRPRESCALER_DIV2: ETRP frequency divided by 2.
  - TIM_ETRPRESCALER_DIV4: ETRP frequency divided by 4.
  - TIM_ETRPRESCALER_DIV8: ETRP frequency divided by 8.
- **TIM_ExtTRGPolarity:** The external Trigger Polarity. This parameter can be one of the following values:
  - TIM_ETRPOLARITY_INVERTED: active low or falling edge active.
  - TIM_ETRPOLARITY_NONINVERTED: active high or rising edge active.
- **ExtTRGFilter:** External Trigger Filter. This parameter must be a value between 0x00 and 0x0F

Return values

- **None:**

TIM_DMADelayPulseHalfCplt

Function name

void TIM_DMADelayPulseHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Delay Pulse half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMAError

Function name

void TIM_DMAError (DMA_HandleTypeDef * hdma)

Function description

TIM DMA error callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureCplt

Function name

void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_DMACaptureHalfCplt

Function name

void TIM_DMACaptureHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Capture half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIM_CCxChannelCmd

Function name

void TIM_CCxChannelCmd (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ChannelState)

Function description

Enables or disables the TIM Capture Compare Channel x.

Parameters

- **TIMx:** to select the TIM peripheral
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1
  - TIM_CHANNEL_2: TIM Channel 2
  - TIM_CHANNEL_3: TIM Channel 3
  - TIM_CHANNEL_4: TIM Channel 4
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- **ChannelState:** specifies the TIM Channel CCxE bit new state. This parameter can be: TIM_CCx_ENABLE or TIM_CCx_DISABLE.

Return values

- **None:**

TIM_DMA_Start_IT

Function name

HAL_StatusTypeDef TIM_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t src, uint32_t dst, uint32_t length)

Function description

Start the DMA data transfer.

Parameters

- **hdma:** DMA handle
- **src:** : The source memory Buffer address.
- **dst:** : The destination memory Buffer address.
- **length:** : The size of a source block transfer in byte.

Return values

- **HAL:** status

TIM_ResetCallback

Function name

void TIM_ResetCallback (TIM_HandleTypeDef * htim)

Function description

Reset interrupt callbacks to the legacy weak callbacks.

Parameters

- **htim:** pointer to a TIM_HandleTypeDef structure that contains the configuration information for TIM module.

Return values

- **None:**

81.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

81.3.1 TIM

TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event (if none of the break inputs BRK and BRK2 is active)

TIM Auto-Reload Preload

TIM_AUTORELOAD_PRELOAD_DISABLE

TIMx_ARR register is not buffered

TIM_AUTORELOAD_PRELOAD_ENABLE

TIMx_ARR register is buffered

TIM Break2 Input Alternate Function Mode

TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

TIM Break input 2 Enable

TIM_BREAK2_DISABLE

Break input BRK2 is disabled

TIM_BREAK2_ENABLE

Break input BRK2 is enabled

TIM Break Input 2 Polarity

TIM_BREAK2POLARITY_LOW

Break input BRK2 is active low

TIM_BREAK2POLARITY_HIGH

Break input BRK2 is active high

TIM Break Input Alternate Function Mode

TIM_BREAK_AFMODE_INPUT

Break input BRK in input mode

TIM_BREAK_AFMODE_BIDIRECTIONAL

Break input BRK in bidirectional mode

TIM Break Input Enable

TIM_BREAK_ENABLE

Break input BRK is enabled

TIM_BREAK_DISABLE

Break input BRK is disabled

TIM Break Input Polarity

TIM_BREAKPOLARITY_LOW

Break input BRK is active low

TIM_BREAKPOLARITY_HIGH

Break input BRK is active high

TIM Break System

TIM_BREAK_SYSTEM_ECC

Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17/20

TIM_BREAK_SYSTEM_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17/20 Break Input and also the PVDE and PLS bits of the Power Control Interface

TIM_BREAK_SYSTEM_SRAM_PARITY_ERROR

Enables and locks the SRAM_PARITY error signal with Break Input of TIM1/8/15/16/17/20

TIM_BREAK_SYSTEM_LOCKUP

Enables and locks the LOCKUP output of CortexM4 with Break Input of TIM1/8/15/16/17/20

CCx DMA request selection

TIM_CCDMAREQUEST_CC

CCx DMA request sent when capture or compare match event occurs

TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

TIM Channel

TIM_CHANNEL_1

Capture/compare channel 1 identifier

TIM_CHANNEL_2

Capture/compare channel 2 identifier

TIM_CHANNEL_3

Capture/compare channel 3 identifier

TIM_CHANNEL_4

Capture/compare channel 4 identifier

TIM_CHANNEL_5

Compare channel 5 identifier

TIM_CHANNEL_6

Compare channel 6 identifier

TIM_CHANNEL_ALL

Global Capture/compare channel identifier

TIM Clear Input Polarity

TIM_CLEARINPUTPOLARITY_INVERTED

Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED

Polarity for ETRx pin

TIM Clear Input Prescaler

TIM_CLEARINPUTPRESCALER_DIV1

No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Clear Input Source

TIM_CLEARINPUTSOURCE_NONE

OCREF_CLR is disabled

TIM_CLEARINPUTSOURCE_ETR

OCREF_CLR is connected to ETRF input

TIM_CLEARINPUTSOURCE_COMP1

OCREF_CLR_INT is connected to COMP1 output

TIM_CLEARINPUTSOURCE_COMP2

OCREF_CLR_INT is connected to COMP2 output

TIM Clock Division

TIM_CLOCKDIVISION_DIV1

Clock division: tDTS=tCK_INT

TIM_CLOCKDIVISION_DIV2

Clock division: tDTS=2*tCK_INT

TIM_CLOCKDIVISION_DIV4

Clock division: tDTS=4*tCK_INT

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED

Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_FALLING

Polarity for TIx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE

Polarity for TIx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1

No prescaler is used

TIM_CLOCKPRESCALER_DIV2

Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4

Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8

Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_INTERNAL

Internal clock source

TIM_CLOCKSOURCE_ETRMODE1

External clock source mode 1 (ETRF)

TIM_CLOCKSOURCE_ETRMODE2

External clock source mode 2

TIM_CLOCKSOURCE_TI1ED

External clock source mode 1 (TTI1FP1 + edge detect.)

TIM_CLOCKSOURCE_TI1

External clock source mode 1 (TTI1FP1)

TIM_CLOCKSOURCE_TI2

External clock source mode 1 (TTI2FP2)

TIM_CLOCKSOURCE_ITR0

External clock source mode 1 (ITR0)

TIM_CLOCKSOURCE_ITR1

External clock source mode 1 (ITR1)

TIM_CLOCKSOURCE_ITR2

External clock source mode 1 (ITR2)

TIM_CLOCKSOURCE_ITR3

External clock source mode 1 (ITR3)

TIM_CLOCKSOURCE_ITR4

External clock source mode 1 (ITR4)

TIM_CLOCKSOURCE_ITR5

External clock source mode 1 (ITR5)

TIM_CLOCKSOURCE_ITR6

External clock source mode 1 (ITR6)

TIM_CLOCKSOURCE_ITR7

External clock source mode 1 (ITR7)

TIM_CLOCKSOURCE_ITR8

External clock source mode 1 (ITR8)

TIM_CLOCKSOURCE_ITR11

External clock source mode 1 (ITR11)

TIM Commutation Source

TIM_COMMUTATION_TRGI

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit or when an rising edge occurs on trigger input

TIM_COMMUTATION_SOFTWARE

When Capture/compare control bits are preloaded, they are updated by setting the COMG bit

TIM Counter Mode

TIM_COUNTERMODE_UP

Counter used as up-counter

TIM_COUNTERMODE_DOWN

Counter used as down-counter

TIM_COUNTERMODE_CENTERALIGNED1

Center-aligned mode 1

TIM_COUNTERMODE_CENTERALIGNED2

Center-aligned mode 2

TIM_COUNTERMODE_CENTERALIGNED3

Center-aligned mode 3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT

TIM_DMABASE_PSC

TIM_DMABASE_ARR

TIM_DMABASE_RCR

TIM_DMABASE_CCR1

TIM_DMABASE_CCR2

TIM_DMABASE_CCR3

TIM_DMABASE_CCR4

TIM_DMABASE_BDTR

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_CCMR3

TIM_DMABASE_DTR2

TIM_DMABASE_ECR

TIM_DMABASE_TISEL

TIM_DMABASE_AF1

TIM_DMABASE_AF2

TIM_DMABASE_OR1

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER

The transfer is done to 1 register starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_2TRANSFERS

The transfer is done to 2 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_3TRANSFERS

The transfer is done to 3 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_4TRANSFERS

The transfer is done to 4 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_5TRANSFERS

The transfer is done to 5 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_6TRANSFERS

The transfer is done to 6 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_7TRANSFERS

The transfer is done to 7 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_8TRANSFERS

The transfer is done to 8 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_9TRANSFERS

The transfer is done to 9 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_10TRANSFERS

The transfer is done to 10 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_11TRANSFERS

The transfer is done to 11 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_12TRANSFERS

The transfer is done to 12 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_13TRANSFERS

The transfer is done to 13 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_14TRANSFERS

The transfer is done to 14 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_15TRANSFERS

The transfer is done to 15 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_16TRANSFERS

The transfer is done to 16 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_17TRANSFERS

The transfer is done to 17 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_18TRANSFERS

The transfer is done to 18 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_19TRANSFERS

The transfer is done to 19 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_20TRANSFERS

The transfer is done to 20 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_21TRANSFERS

The transfer is done to 21 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_22TRANSFERS

The transfer is done to 22 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_23TRANSFERS

The transfer is done to 23 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_24TRANSFERS

The transfer is done to 24 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_25TRANSFERS

The transfer is done to 25 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM_DMABURSTLENGTH_26TRANSFERS

The transfer is done to 26 registers starting from TIMx_CR1 + TIMx_DCR.DBA

TIM DMA Sources

TIM_DMA_UPDATE

DMA request is triggered by the update event

TIM_DMA_CC1

DMA request is triggered by the capture/compare macth 1 event

TIM_DMA_CC2

DMA request is triggered by the capture/compare macth 2 event event

TIM_DMA_CC3

DMA request is triggered by the capture/compare macth 3 event event

TIM_DMA_CC4

DMA request is triggered by the capture/compare macth 4 event event

TIM_DMA_COM

DMA request is triggered by the commutation event

TIM_DMA_TRIGGER

DMA request is triggered by the trigger event

TIM Encoder Input Polarity

TIM_ENCODERINPUTPOLARITY_RISING

Encoder input with rising edge polarity

TIM_ENCODERINPUTPOLARITY_FALLING

Encoder input with falling edge polarity

TIM Encoder Mode

TIM_ENCODERMODE_TI1

Quadrature encoder mode 1, x2 mode, counts up/down on TI1FP1 edge depending on TI2FP2 level

TIM_ENCODERMODE_TI2

Quadrature encoder mode 2, x2 mode, counts up/down on TI2FP2 edge depending on TI1FP1 level.

TIM_ENCODERMODE_TI12

Quadrature encoder mode 3, x4 mode, counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction, x2 mode

TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED

Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED

Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1

No prescaler is used

TIM_ETRPRESCALER_DIV2

ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4

ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8

ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

Reinitialize the counter and generates an update of the registers

TIM_EVENTSOURCE_CC1

A capture/compare event is generated on channel 1

TIM_EVENTSOURCE_CC2

A capture/compare event is generated on channel 2

TIM_EVENTSOURCE_CC3

A capture/compare event is generated on channel 3

TIM_EVENTSOURCE_CC4

A capture/compare event is generated on channel 4

TIM_EVENTSOURCE_COM

A commutation event is generated

TIM_EVENTSOURCE_TRIGGER

A trigger event is generated

TIM_EVENTSOURCE_BREAK

A break event is generated

TIM_EVENTSOURCE_BREAK2

A break 2 event is generated

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:
- Reset TIM handle state.

Parameters:
- __HANDLE__: TIM handle.

Return value:
- None

__HAL_TIM_ENABLE

Description:
- Enable the TIM peripheral.

Parameters:
- __HANDLE__: TIM handle

Return value:
- None

__HAL_TIM_MOE_ENABLE

Description:
- Enable the TIM main Output.

Parameters:
- __HANDLE__: TIM handle

Return value:
- None

__HAL_TIM_DISABLE

Description:
- Disable the TIM peripheral.

Parameters:
- __HANDLE__: TIM handle

Return value:
- None

__HAL_TIM_MOE_DISABLE

Description:
- Disable the TIM main Output.

Parameters:
- __HANDLE__: TIM handle

Return value:
- None

Notes:
- The Main Output Enable of a timer instance is disabled only if all the CCx and CCxN channels have been disabled

__HAL_TIM_MOE_DISABLE_UNCONDITIONALLY

Description:
- Disable the TIM main Output.

Parameters:
- __HANDLE__: TIM handle

Return value:
- None

Notes:
- The Main Output Enable of a timer instance is disabled unconditionally

__HAL_TIM_ENABLE_IT

Description:
- Enable the specified TIM interrupt.

Parameters:
- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:
- None

__HAL_TIM_DISABLE_IT

Description:

- Disable the specified TIM interrupt.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __INTERRUPT__: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_ENABLE_DMA

Description:

- Enable the specified DMA request.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to enable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA

Description:
- Disable the specified DMA request.

Parameters:
- __HANDLE__: specifies the TIM Handle.
- __DMA__: specifies the TIM DMA request to disable. This parameter can be one of the following values:
  - TIM_DMA_UPDATE: Update DMA request
  - TIM_DMA_CC1: Capture/Compare 1 DMA request
  - TIM_DMA_CC2: Capture/Compare 2 DMA request
  - TIM_DMA_CC3: Capture/Compare 3 DMA request
  - TIM_DMA_CC4: Capture/Compare 4 DMA request
  - TIM_DMA_COM: Commutation DMA request
  - TIM_DMA_TRIGGER: Trigger DMA request

Return value:
- None

__HAL_TIM_GET_FLAG

Description:
- Check whether the specified TIM interrupt flag is set or not.

Parameters:
- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:
- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_CLEAR_FLAG

Description:

- Clear the specified TIM interrupt flag.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __FLAG__: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
  - TIM_FLAG_UPDATE: Update interrupt flag
  - TIM_FLAG_CC1: Capture/Compare 1 interrupt flag
  - TIM_FLAG_CC2: Capture/Compare 2 interrupt flag
  - TIM_FLAG_CC3: Capture/Compare 3 interrupt flag
  - TIM_FLAG_CC4: Capture/Compare 4 interrupt flag
  - TIM_FLAG_CC5: Compare 5 interrupt flag
  - TIM_FLAG_CC6: Compare 6 interrupt flag
  - TIM_FLAG_COM: Commutation interrupt flag
  - TIM_FLAG_TRIGGER: Trigger interrupt flag
  - TIM_FLAG_BREAK: Break interrupt flag
  - TIM_FLAG_BREAK2: Break 2 interrupt flag
  - TIM_FLAG_SYSTEM_BREAK: System Break interrupt flag
  - TIM_FLAG_CC1OF: Capture/Compare 1 overcapture flag
  - TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
  - TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
  - TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag
  - TIM_FLAG_IDX: Index interrupt flag
  - TIM_FLAG_DIR: Direction change interrupt flag
  - TIM_FLAG_IERR: Index error interrupt flag
  - TIM_FLAG_TERR: Transition error interrupt flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE

Description:

- Check whether the specified TIM interrupt source is enabled or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check. This parameter can be one of the following values:
  - TIM_IT_UPDATE: Update interrupt
  - TIM_IT_CC1: Capture/Compare 1 interrupt
  - TIM_IT_CC2: Capture/Compare 2 interrupt
  - TIM_IT_CC3: Capture/Compare 3 interrupt
  - TIM_IT_CC4: Capture/Compare 4 interrupt
  - TIM_IT_COM: Commutation interrupt
  - TIM_IT_TRIGGER: Trigger interrupt
  - TIM_IT_BREAK: Break interrupt
  - TIM_IT_IDX: Index interrupt
  - TIM_IT_DIR: Direction change interrupt
  - TIM_IT_IERR: Index error interrupt
  - TIM_IT_TERR: Transition error interrupt

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT

Description:

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  – TIM_IT_UPDATE: Update interrupt
  – TIM_IT_CC1: Capture/Compare 1 interrupt
  – TIM_IT_CC2: Capture/Compare 2 interrupt
  – TIM_IT_CC3: Capture/Compare 3 interrupt
  – TIM_IT_CC4: Capture/Compare 4 interrupt
  – TIM_IT_COM: Commutation interrupt
  – TIM_IT_TRIGGER: Trigger interrupt
  – TIM_IT_BREAK: Break interrupt
  – TIM_IT_IDX: Index interrupt
  – TIM_IT_DIR: Direction change interrupt
  – TIM_IT_IERR: Index error interrupt
  – TIM_IT_TERR: Transition error interrupt

Return value:

- None

__HAL_TIM_UIFREMAP_ENABLE

Description:

- Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

Notes:

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

__HAL_TIM_UIFREMAP_DISABLE

Description:

- Disable update interrupt flag (UIF) remapping.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None: mode.

__HAL_TIM_GET_UIFCPY

Description:

- Get update interrupt flag (UIF) copy status.

Parameters:

- __COUNTER__: Counter value.

Return value:

- The: state of UIFCPY (TRUE or FALSE). mode.

__HAL_TIM_IS_TIM_COUNTING_DOWN

Description:
- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:
- __HANDLE__: TIM handle.

Return value:
- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:
- This macro is particularly useful to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER

Description:
- Set the TIM Prescaler on runtime.

Parameters:
- __HANDLE__: TIM handle.
- __PRESC__: specifies the Prescaler new value.

Return value:
- None

__HAL_TIM_SET_COUNTER

Description:
- Set the TIM Counter Register value on runtime.

Parameters:
- __HANDLE__: TIM handle.
- __COUNTER__: specifies the Counter register new value.

Return value:
- None

__HAL_TIM_GET_COUNTER

Description:
- Get the TIM Counter Register value on runtime.

Parameters:
- __HANDLE__: TIM handle.

Return value:
- 16-bit: or 32-bit value of the timer counter register (TIMx_CNT)

__HAL_TIM_SET_AUTORELOAD

Description:
- Set the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:
- __HANDLE__: TIM handle.
- __AUTORELOAD__: specifies the Counter register new value.

Return value:
- None

__HAL_TIM_GET_AUTORELOAD

Description:

- Get the TIM Autoreload Register value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- 16-bit: or 32-bit value of the timer auto-reload register(TIMx_ARR)

__HAL_TIM_SET_CLOCKDIVISION

Description:

- Set the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- __HANDLE__: TIM handle.
- __CKD__: specifies the clock division value. This parameter can be one of the following value:
  – TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  – TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  – TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

Return value:

- None

__HAL_TIM_GET_CLOCKDIVISION

Description:

- Get the TIM Clock Division value on runtime.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- The: clock division can be one of the following values:
  – TIM_CLOCKDIVISION_DIV1: tDTS=tCK_INT
  – TIM_CLOCKDIVISION_DIV2: tDTS=2*tCK_INT
  – TIM_CLOCKDIVISION_DIV4: tDTS=4*tCK_INT

__HAL_TIM_SET_ICPRESCALER

Description:

- Set the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
- __ICPSC__: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
  – TIM_ICPSC_DIV1: no prescaler
  – TIM_ICPSC_DIV2: capture is done once every 2 events
  – TIM_ICPSC_DIV4: capture is done once every 4 events
  – TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

__HAL_TIM_GET_ICPRESCALER

Description:
- Get the TIM Input Capture prescaler on runtime.

Parameters:
- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: get input capture 1 prescaler value
  - TIM_CHANNEL_2: get input capture 2 prescaler value
  - TIM_CHANNEL_3: get input capture 3 prescaler value
  - TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:
- The: input capture prescaler can be one of the following values:
  - TIM_ICPSC_DIV1: no prescaler
  - TIM_ICPSC_DIV2: capture is done once every 2 events
  - TIM_ICPSC_DIV4: capture is done once every 4 events
  - TIM_ICPSC_DIV8: capture is done once every 8 events

__HAL_TIM_SET_COMPARE

Description:
- Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:
- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected
- __COMPARE__: specifies the Capture Compare register new value.

Return value:
- None

__HAL_TIM_GET_COMPARE

Description:
- Get the TIM Capture Compare Register value on runtime.

Parameters:
- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channel associated with the capture compare register This parameter can be one of the following values:
  - TIM_CHANNEL_1: get capture/compare 1 register value
  - TIM_CHANNEL_2: get capture/compare 2 register value
  - TIM_CHANNEL_3: get capture/compare 3 register value
  - TIM_CHANNEL_4: get capture/compare 4 register value
  - TIM_CHANNEL_5: get capture/compare 5 register value
  - TIM_CHANNEL_6: get capture/compare 6 register value

Return value:
- 16-bit: or 32-bit value of the capture/compare register (TIMx_CCRy)

__HAL_TIM_ENABLE_OCxPRELOAD

Description:

- Set the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_DISABLE_OCxPRELOAD

Description:

- Reset the TIM Output compare preload.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

__HAL_TIM_ENABLE_OCxFAST

Description:

- Enable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is enabled an active edge on the trigger input acts like a compare match on CCx output. Delay to sample the trigger input and to activate CCx output is reduced to 3 clock cycles. Fast mode acts only if the channel is configured in PWM1 or PWM2 mode.

__HAL_TIM_DISABLE_OCxFAST

Description:

- Disable fast mode for a given channel.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected
  - TIM_CHANNEL_5: TIM Channel 5 selected
  - TIM_CHANNEL_6: TIM Channel 6 selected

Return value:

- None

Notes:

- When fast mode is disabled CCx output behaves normally depending on counter and CCRx values even when the trigger is ON. The minimum delay to activate CCx output when an active edge occurs on the trigger input is 5 clock cycles.

__HAL_TIM_URS_ENABLE

Description:

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE

Description:

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the URS bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): _ Counter overflow underflow _ Setting the UG bit _ Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY

Description:

- Set the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
  – TIM_INPUTCHANNELPOLARITY_RISING: Rising Edge
  – TIM_INPUTCHANNELPOLARITY_FALLING: Falling Edge
  – TIM_INPUTCHANNELPOLARITY_BOTHEDGE: Rising and Falling Edge

Return value:

- None

__HAL_TIM_SELECT_CCDMAREQUEST

Description:

- Select the Capture/compare DMA request source.

Parameters:

- __HANDLE__: specifies the TIM Handle.
- __CCDMA__: specifies Capture/compare DMA request source This parameter can be one of the following values:
  – TIM_CCDMAREQUEST_CC: CCx DMA request generated on Capture/Compare event
  – TIM_CCDMAREQUEST_UPDATE: CCx DMA request generated on Update event

Return value:

- None

TIM Flag Definition

TIM_FLAG_UPDATE

Update interrupt flag

TIM_FLAG_CC1

Capture/Compare 1 interrupt flag

TIM_FLAG_CC2

Capture/Compare 2 interrupt flag

TIM_FLAG_CC3

Capture/Compare 3 interrupt flag

TIM_FLAG_CC4

Capture/Compare 4 interrupt flag

TIM_FLAG_CC5

Capture/Compare 5 interrupt flag

TIM_FLAG_CC6

Capture/Compare 6 interrupt flag

TIM_FLAG_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

Break interrupt flag

TIM_FLAG_BREAK2

Break 2 interrupt flag

TIM_FLAG_SYSTEM_BREAK

System Break interrupt flag

TIM_FLAG_CC1OF

Capture 1 overcapture flag

TIM_FLAG_CC2OF

Capture 2 overcapture flag

TIM_FLAG_CC3OF

Capture 3 overcapture flag

TIM_FLAG_CC4OF

Capture 4 overcapture flag

TIM_FLAG_IDX

Encoder index flag

TIM_FLAG_DIR

Direction change flag

TIM_FLAG_IERR

Index error flag

TIM_FLAG_TERR

Transition error flag

TIM Group Channel 5 and Channel 1, 2 or 3

TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

Capture triggered by rising edge on timer input

TIM_ICPOLARITY_FALLING

Capture triggered by falling edge on timer input

TIM_ICPOLARITY_BOTHEDGE

Capture triggered by both rising and falling edges on timer input

TIM Input Capture Prescaler

TIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2

Capture performed once every 2 events

TIM_ICPSC_DIV4

Capture performed once every 4 events

TIM_ICPSC_DIV8

Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively

TIM_ICSELECTION_INDIRECTTI

TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively

TIM_ICSELECTION_TRC

TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel polarity

TIM_INPUTCHANNELPOLARITY_RISING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_FALLING

Polarity for TIx source

TIM_INPUTCHANNELPOLARITY_BOTHEDGE

Polarity for TIx source

TIM interrupt Definition

TIM_IT_UPDATE

Update interrupt

TIM_IT_CC1

Capture/Compare 1 interrupt

TIM_IT_CC2

Capture/Compare 2 interrupt

TIM_IT_CC3

Capture/Compare 3 interrupt

TIM_IT_CC4

Capture/Compare 4 interrupt

TIM_IT_COM

Commutation interrupt

TIM_IT_TRIGGER

Trigger interrupt

TIM_IT_BREAK

Break interrupt

TIM_IT_IDX

Index interrupt

TIM_IT_DIR

Direction change interrupt

TIM_IT_IERR

Index error interrupt

TIM_IT_TERR

Transition error interrupt

TIM Lock level

TIM_LOCKLEVEL_OFF

LOCK OFF

TIM_LOCKLEVEL_1

LOCK Level 1

TIM_LOCKLEVEL_2

LOCK Level 2

TIM_LOCKLEVEL_3

LOCK Level 3

TIM Master Mode Selection

TIM_TRGO_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO)

TIM_TRGO_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO)

TIM_TRGO_UPDATE

Update event is used as trigger output (TRGO)

TIM_TRGO_OC1

Capture or a compare match 1 is used as trigger output (TRGO)

TIM_TRGO_OC1REF

OC1REF signal is used as trigger output (TRGO)

TIM_TRGO_OC2REF

OC2REF signal is used as trigger output(TRGO)

TIM_TRGO_OC3REF

OC3REF signal is used as trigger output(TRGO)

TIM_TRGO_OC4REF

OC4REF signal is used as trigger output(TRGO)

TIM_TRGO_ENCODER_CLK

Encoder clock is used as trigger output(TRGO)

TIM Master Mode Selection 2 (TRGO2)

TIM_TRGO2_RESET

TIMx_EGR.UG bit is used as trigger output (TRGO2)

TIM_TRGO2_ENABLE

TIMx_CR1.CEN bit is used as trigger output (TRGO2)

TIM_TRGO2_UPDATE

Update event is used as trigger output (TRGO2)

TIM_TRGO2_OC1

Capture or a compare match 1 is used as trigger output (TRGO2)

TIM_TRGO2_OC1REF

OC1REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC2REF

OC2REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC3REF

OC3REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF

OC4REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC5REF

OC5REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC6REF

OC6REF signal is used as trigger output (TRGO2)

TIM_TRGO2_OC4REF_RISINGFALLING

OC4REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC6REF_RISINGFALLING

OC6REF rising or falling edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_RISING

OC4REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC4REF_RISING_OC6REF_FALLING

OC4REF rising or OC6REF falling edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_RISING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM_TRGO2_OC5REF_RISING_OC6REF_FALLING

OC5REF or OC6REF rising edges generate pulses on TRGO2

TIM Master/Slave Mode

TIM_MASTERSLAVEMODE_ENABLE

No action

TIM_MASTERSLAVEMODE_DISABLE

Master/slave mode is selected

TIM One Pulse Mode

TIM_OPMODE_SINGLE

Counter stops counting at the next update event

TIM_OPMODE_REPETITIVE

Counter is not stopped at update event

TIM OSSI OffState Selection for Idle mode state

TIM_OSSI_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSI_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM OSSR OffState Selection for Run mode state

TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled (still controlled by the timer)

TIM_OSSR_DISABLE

When inactive, OC/OCN outputs are disabled (not controlled any longer by the timer)

TIM Output Compare and PWM Modes

TIM_OCMODE_TIMING

Frozen

TIM_OCMODE_ACTIVE

Set channel to active level on match

TIM_OCMODE_INACTIVE

Set channel to inactive level on match

TIM_OCMODE_TOGGLE

Toggle

TIM_OCMODE_PWM1

PWM mode 1

TIM_OCMODE_PWM2

PWM mode 2

TIM_OCMODE_FORCED_ACTIVE

Force active level

TIM_OCMODE_FORCED_INACTIVE

Force inactive level

TIM_OCMODE_RETRIGERRABLE_OPM1

Retrigerrable OPM mode 1

TIM_OCMODE_RETRIGERRABLE_OPM2

Retrigerrable OPM mode 2

TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

TIM_OCMODE_ASYMMETRIC_PWM1

Asymmetric PWM mode 1

TIM_OCMODE_ASYMMETRIC_PWM2

Asymmetric PWM mode 2

TIM_OCMODE_PULSE_ON_COMPARE

Pulse on compare (CH3&CH4 only)

TIM_OCMODE_DIRECTION_OUTPUT

Direction output (CH3&CH4 only)

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

Output Idle state: OCx=1 when MOE=0

TIM_OCIDLESTATE_RESET

Output Idle state: OCx=0 when MOE=0

TIM Complementary Output Compare Idle State

TIM_OCNIDLESTATE_SET

Complementary output Idle state: OCxN=1 when MOE=0

TIM_OCNIDLESTATE_RESET

Complementary output Idle state: OCxN=0 when MOE=0

TIM Complementary Output Compare Polarity

TIM_OCNPOLARITY_HIGH

Capture/Compare complementary output polarity

TIM_OCNPOLARITY_LOW

Capture/Compare complementary output polarity

TIM Complementary Output Compare State

TIM_OUTPUTNSTATE_DISABLE

OCxN is disabled

TIM_OUTPUTNSTATE_ENABLE

OCxN is enabled

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

Capture/Compare output polarity

TIM_OCPOLARITY_LOW

Capture/Compare output polarity

TIM Output Compare State

TIM_OUTPUTSTATE_DISABLE

Capture/Compare 1 output disabled

TIM_OUTPUTSTATE_ENABLE

Capture/Compare 1 output enabled

TIM Output Fast State

TIM_OCFAST_DISABLE

Output Compare fast disable

TIM_OCFAST_ENABLE

Output Compare fast enable

TIM Slave mode

TIM_SLAVEMODE_DISABLE

Slave mode disabled

TIM_SLAVEMODE_RESET

Reset Mode

TIM_SLAVEMODE_GATED

Gated Mode

TIM_SLAVEMODE_TRIGGER

Trigger Mode

TIM_SLAVEMODE_EXTERNAL1

External Clock Mode 1

TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode

TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

The TIMx_CH1 pin is connected to TI1 input

TIM_TI1SELECTION_XORCOMBINATION

The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED

Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING

Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE

Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1

No prescaler is used

TIM_TRIGGERPRESCALER_DIV2

Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4

Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8

Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

Internal Trigger 0 (ITR0)

TIM_TS_ITR1

Internal Trigger 1 (ITR1)

TIM_TS_ITR2

Internal Trigger 2 (ITR2)

TIM_TS_ITR3

Internal Trigger 3 (ITR3)

TIM_TS_ITR4

Internal Trigger 4 (ITR4)

TIM_TS_ITR5

Internal Trigger 5 (ITR5)

TIM_TS_ITR6

Internal Trigger 6 (ITR6)

TIM_TS_ITR7

Internal Trigger 7 (ITR7)

TIM_TS_ITR8

Internal Trigger 8 (ITR8)

TIM_TS_ITR11

Internal Trigger 11 (ITR11)

TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED)

TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1)

TIM_TS_TI2FP2

Filtered Timer Input 2 (TI2FP2)

TIM_TS_ETRF

Filtered External Trigger input (ETRF)

TIM_TS_NONE

No trigger selected

TIM Update Interrupt Flag Remap

TIM_UIFREMAP_DISABLE

Update interrupt flag remap disabled

TIM_UIFREMAP_ENABLE

Update interrupt flag remap enabled

82 HAL TIM Extension Driver

82.1 TIMEx Firmware driver registers structures

82.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32u5xx_hal_tim_ex.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t Commutation_Delay***

Field Documentation

- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Polarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_Input_Capture_Polarity***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Prescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_Input_Capture_Prescaler***
- ***uint32_t TIM_HallSensor_InitTypeDef::IC1Filter***
  Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIM_HallSensor_InitTypeDef::Commutation_Delay***
  Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

82.1.2 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32u5xx_hal_tim_ex.h

Data Fields

- ***uint32_t Source***
- ***uint32_t Enable***
- ***uint32_t Polarity***

Field Documentation

- ***uint32_t TIMEx_BreakInputConfigTypeDef::Source***
  Specifies the source of the timer break input. This parameter can be a value of ***TIMEx_Break_Input_Source***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Enable***
  Specifies whether or not the break input source is enabled. This parameter can be a value of ***TIMEx_Break_Input_Source_Enable***
- ***uint32_t TIMEx_BreakInputConfigTypeDef::Polarity***
  Specifies the break input source polarity. This parameter can be a value of ***TIMEx_Break_Input_Source_Polarity***

82.1.3 TIMEx_EncoderIndexConfigTypeDef

TIMEx_EncoderIndexConfigTypeDef is defined in the stm32u5xx_hal_tim_ex.h

Data Fields

- ***uint32_t Polarity***
- ***uint32_t Prescaler***
- ***uint32_t Filter***
- ***uint32_t Blanking***
- ***FunctionalState FirstIndexEnable***
- ***uint32_t Position***
- ***uint32_t Direction***

Field Documentation

- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Polarity***
  TIM Encoder index polarity.This parameter can be a value of ***TIMEx_Encoder_Index_Polarity***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Prescaler***
  TIM Encoder index prescaler.This parameter can be a value of ***TIMEx_Encoder_Index_Prescaler***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Filter***
  TIM Encoder index filter.This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Blanking***
  Specifies whether or not the encoder index event is conditioned by TI3 or TI4 input.This parameter can be a value of ***TIMEx_Encoder_Index_Blanking***
- ***FunctionalState TIMEx_EncoderIndexConfigTypeDef::FirstIndexEnable***
  Specifies whether or not the encoder first index is enabled.This parameter value can be ENABLE or DISABLE.
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Position***
  Specifies in which AB input configuration the index event resets the counter.This parameter can be a value of ***TIMEx_Encoder_Index_Position***
- ***uint32_t TIMEx_EncoderIndexConfigTypeDef::Direction***
  Specifies in which counter direction the index event resets the counter.This parameter can be a value of ***TIMEx_Encoder_Index_Direction***

82.2 TIMEx Firmware driver API description

The following section lists the various functions of the TIMEx library.

82.2.1 TIMER Extended features

The Timer Extended features include:

1. Complementary outputs with programmable dead-time for :
   – Output Compare
   – PWM generation (Edge and Center-aligned Mode)
   – One-pulse mode output
2. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
3. Break input to put the timer output signals in reset state or in a known state.
4. Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
5. In case of Pulse on compare, configure pulse length and delay
6. Encoder index configuration

82.2.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending on the selected feature:
   – Hall Sensor output : HAL_TIMEx_HallSensor_MspInit()
2. Initialize the TIM low level resources :
   a. Enable the TIM interface clock using __HAL_RCC_TIMx_CLK_ENABLE();
   b. TIM pins configuration
      ◦ Enable the clock for the TIM GPIOs using the following function: __HAL_RCC_GPIOx_CLK_ENABLE();
      ◦ Configure these TIM pins in Alternate function mode using HAL_GPIO_Init();
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: HAL_TIM_ConfigClockSource, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the initialization function of this driver:
   – HAL_TIMEx_HallSensor_Init() and HAL_TIMEx_ConfigCommutEvent(): to use the Timer Hall Sensor Interface and the commutation event with the corresponding Interrupt and DMA request if needed (Note that One Timer is used to interface with the Hall sensor Interface and another Timer should be used to use the commutation event).

5. In case of Pulse On Compare:
   – HAL_TIMEx_OC_ConfigPulseOnCompare(): to configure pulse width and prescaler
6. Activate the TIM peripheral using one of the start functions:
   – Complementary Output Compare : HAL_TIMEx_OCN_Start(), HAL_TIMEx_OCN_Start_DMA(), HAL_TIMEx_OCN_Start_IT()
   – Complementary PWM generation : HAL_TIMEx_PWMN_Start(), HAL_TIMEx_PWMN_Start_DMA(), HAL_TIMEx_PWMN_Start_IT()
   – Complementary One-pulse mode output : HAL_TIMEx_OnePulseN_Start(), HAL_TIMEx_OnePulseN_Start_IT()
   – Hall Sensor output : HAL_TIMEx_HallSensor_Start(), HAL_TIMEx_HallSensor_Start_DMA(), HAL_TIMEx_HallSensor_Start_IT().

82.2.3 Timer Hall Sensor functions

This section provides functions allowing to:
- Initialize and configure TIM HAL Sensor.
- De-initialize TIM HAL Sensor.
- Start the Hall Sensor Interface.
- Stop the Hall Sensor Interface.
- Start the Hall Sensor Interface and enable interrupts.
- Stop the Hall Sensor Interface and disable interrupts.
- Start the Hall Sensor Interface and enable DMA transfers.
- Stop the Hall Sensor Interface and disable DMA transfers.

This section contains the following APIs:
- *HAL_TIMEx_HallSensor_Init()*
- *HAL_TIMEx_HallSensor_DeInit()*
- *HAL_TIMEx_HallSensor_MspInit()*
- *HAL_TIMEx_HallSensor_MspDeInit()*
- *HAL_TIMEx_HallSensor_Start()*
- *HAL_TIMEx_HallSensor_Stop()*
- *HAL_TIMEx_HallSensor_Start_IT()*
- *HAL_TIMEx_HallSensor_Stop_IT()*
- *HAL_TIMEx_HallSensor_Start_DMA()*
- *HAL_TIMEx_HallSensor_Stop_DMA()*

82.2.4 Timer Complementary Output Compare functions

This section provides functions allowing to:
- Start the Complementary Output Compare/PWM.
- Stop the Complementary Output Compare/PWM.
- Start the Complementary Output Compare/PWM and enable interrupts.
- Stop the Complementary Output Compare/PWM and disable interrupts.
- Start the Complementary Output Compare/PWM and enable DMA transfers.
- Stop the Complementary Output Compare/PWM and disable DMA transfers.

This section contains the following APIs:
- *HAL_TIMEx_OCN_Start()*
- *HAL_TIMEx_OCN_Stop()*
- *HAL_TIMEx_OCN_Start_IT()*
- *HAL_TIMEx_OCN_Stop_IT()*
- *HAL_TIMEx_OCN_Start_DMA()*
- *HAL_TIMEx_OCN_Stop_DMA()*

82.2.5 Timer Complementary PWM functions

This section provides functions allowing to:

- Start the Complementary PWM.
- Stop the Complementary PWM.
- Start the Complementary PWM and enable interrupts.
- Stop the Complementary PWM and disable interrupts.
- Start the Complementary PWM and enable DMA transfers.
- Stop the Complementary PWM and disable DMA transfers.
- Start the Complementary Input Capture measurement.
- Stop the Complementary Input Capture.
- Start the Complementary Input Capture and enable interrupts.
- Stop the Complementary Input Capture and disable interrupts.
- Start the Complementary Input Capture and enable DMA transfers.
- Stop the Complementary Input Capture and disable DMA transfers.
- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- *HAL_TIMEx_PWMN_Start()*
- *HAL_TIMEx_PWMN_Stop()*
- *HAL_TIMEx_PWMN_Start_IT()*
- *HAL_TIMEx_PWMN_Stop_IT()*
- *HAL_TIMEx_PWMN_Start_DMA()*
- *HAL_TIMEx_PWMN_Stop_DMA()*

82.2.6 Timer Complementary One Pulse functions

This section provides functions allowing to:

- Start the Complementary One Pulse generation.
- Stop the Complementary One Pulse.
- Start the Complementary One Pulse and enable interrupts.
- Stop the Complementary One Pulse and disable interrupts.

This section contains the following APIs:

- *HAL_TIMEx_OnePulseN_Start()*
- *HAL_TIMEx_OnePulseN_Stop()*
- *HAL_TIMEx_OnePulseN_Start_IT()*
- *HAL_TIMEx_OnePulseN_Stop_IT()*

82.2.7 Peripheral Control functions

This section provides functions allowing to:

- Configure the commutation event in case of use of the Hall sensor interface.
- Configure Output channels for OC and PWM mode.
- Configure Complementary channels, break features and dead time.
- Configure Master synchronization.
- Configure timer remapping capabilities.
- Select timer input source.
- Enable or disable channel grouping.
- Configure Pulse on compare.
- Configure Encoder index.

This section contains the following APIs:
- *HAL_TIMEx_ConfigCommutEvent()*
- *HAL_TIMEx_ConfigCommutEvent_IT()*
- *HAL_TIMEx_ConfigCommutEvent_DMA()*
- *HAL_TIMEx_MasterConfigSynchronization()*
- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_ConfigBreakInput()*
- *HAL_TIMEx_RemapConfig()*
- *HAL_TIMEx_TISelection()*
- *HAL_TIMEx_GroupChannel5()*
- *HAL_TIMEx_DisarmBreakInput()*
- *HAL_TIMEx_ReArmBreakInput()*
- *HAL_TIMEx_DitheringEnable()*
- *HAL_TIMEx_DitheringDisable()*
- *HAL_TIMEx_OC_ConfigPulseOnCompare()*
- *HAL_TIMEx_ConfigSlaveModePreload()*
- *HAL_TIMEx_EnableSlaveModePreload()*
- *HAL_TIMEx_DisableSlaveModePreload()*
- *HAL_TIMEx_EnableDeadTimePreload()*
- *HAL_TIMEx_DisableDeadTimePreload()*
- *HAL_TIMEx_ConfigDeadTime()*
- *HAL_TIMEx_ConfigAsymmetricalDeadTime()*
- *HAL_TIMEx_EnableAsymmetricalDeadTime()*
- *HAL_TIMEx_DisableAsymmetricalDeadTime()*
- *HAL_TIMEx_ConfigEncoderIndex()*
- *HAL_TIMEx_EnableEncoderIndex()*
- *HAL_TIMEx_DisableEncoderIndex()*
- *HAL_TIMEx_EnableEncoderFirstIndex()*
- *HAL_TIMEx_DisableEncoderFirstIndex()*
- *HAL_TIMEx_EnableHSE32()*
- *HAL_TIMEx_DisableHSE32()*

82.2.8 Extended Callbacks functions

This section provides Extended TIM callback functions:
- Timer Commutation callback
- Timer Break callback

This section contains the following APIs:
- *HAL_TIMEx_CommutCallback()*
- *HAL_TIMEx_CommutHalfCpltCallback()*
- *HAL_TIMEx_BreakCallback()*
- *HAL_TIMEx_Break2Callback()*
- *HAL_TIMEx_EncoderIndexCallback()*
- *HAL_TIMEx_DirectionChangeCallback()*
- *HAL_TIMEx_IndexErrorCallback()*
- *HAL_TIMEx_TransitionErrorCallback()*

82.2.9 Extended Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:
- *HAL_TIMEx_HallSensor_GetState()*

- *HAL_TIMEx_GetChannelNState()*

82.2.10 Detailed description of functions

HAL_TIMEx_HallSensor_Init

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Init (TIM_HandleTypeDef * htim, const TIM_HallSensor_InitTypeDef * sConfig)

Function description

Initializes the TIM Hall Sensor Interface and initialize the associated handle.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **sConfig:** TIM Hall Sensor configuration structure

Return values

- **HAL:** status

Notes

- When the timer instance is initialized in Hall Sensor Interface mode, timer channels 1 and channel 2 are reserved and cannot be used for other purpose.

HAL_TIMEx_HallSensor_DeInit

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_DeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes the TIM Hall Sensor interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_MspInit

Function name

void HAL_TIMEx_HallSensor_MspInit (TIM_HandleTypeDef * htim)

Function description

Initializes the TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_MspDeInit

Function name

void HAL_TIMEx_HallSensor_MspDeInit (TIM_HandleTypeDef * htim)

Function description

DeInitializes TIM Hall Sensor MSP.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **None:**

HAL_TIMEx_HallSensor_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall sensor Interface.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_IT (TIM_HandleTypeDef * htim)

Function description

Starts the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_IT (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in interrupt mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Start_DMA (TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle
- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- **HAL:** status

HAL_TIMEx_HallSensor_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_HallSensor_Stop_DMA (TIM_HandleTypeDef * htim)

Function description

Stops the TIM Hall Sensor Interface in DMA mode.

Parameters

- **htim:** TIM Hall Sensor Interface handle

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM OC handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OCN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_OCN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_OCN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM Output Compare signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM Output Compare handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Starts the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  - TIM_CHANNEL_1: TIM Channel 1 selected
  - TIM_CHANNEL_2: TIM Channel 2 selected
  - TIM_CHANNEL_3: TIM Channel 3 selected
  - TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the PWM signal generation in interrupt mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Start_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, const uint32_t * pData, uint16_t Length)

Function description

Starts the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be enabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- **HAL:** status

HAL_TIMEx_PWMN_Stop_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_PWMN_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)

Function description

Stops the TIM PWM signal generation in DMA mode on the complementary output.

Parameters

- **htim:** TIM handle
- **Channel:** TIM Channel to be disabled This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected
  – TIM_CHANNEL_3: TIM Channel 3 selected
  – TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- **HAL:** status

HAL_TIMEx_OnePulseN_Start

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to enable This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Stop

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation on the complementary output.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to disable This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Start_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Starts the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to enable This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_OnePulseN_Stop_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_OnePulseN_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)

Function description

Stops the TIM One Pulse signal generation in interrupt mode on the complementary channel.

Parameters

- **htim:** TIM One Pulse handle
- **OutputChannel:** pulse output channel to disable This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1 selected
  – TIM_CHANNEL_2: TIM Channel 2 selected

Return values

- **HAL:** status

Notes

- OutputChannel must match the pulse output channel chosen when calling HAL_TIM_OnePulse_ConfigChannel().

HAL_TIMEx_ConfigCommutEvent

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_IT

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_IT (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with interrupt.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  – TIM_TS_ITR0: Internal trigger 0 selected
  – TIM_TS_ITR1: Internal trigger 1 selected
  – TIM_TS_ITR2: Internal trigger 2 selected
  – TIM_TS_ITR3: Internal trigger 3 selected
  – TIM_TS_ITR4: Internal trigger 4 selected
  – TIM_TS_ITR5: Internal trigger 5 selected
  – TIM_TS_ITR6: Internal trigger 6 selected
  – TIM_TS_ITR7: Internal trigger 7 selected
  – TIM_TS_ITR8: Internal trigger 8 selected
  – TIM_TS_ITR11: Internal trigger 11 selected
  – TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  – TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  – TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.

HAL_TIMEx_ConfigCommutEvent_DMA

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigCommutEvent_DMA (TIM_HandleTypeDef * htim, uint32_t InputTrigger, uint32_t CommutationSource)

Function description

Configure the TIM commutation event sequence with DMA.

Parameters

- **htim:** TIM handle
- **InputTrigger:** the Internal trigger corresponding to the Timer Interfacing with the Hall sensor This parameter can be one of the following values:
  - TIM_TS_ITR0: Internal trigger 0 selected
  - TIM_TS_ITR1: Internal trigger 1 selected
  - TIM_TS_ITR2: Internal trigger 2 selected
  - TIM_TS_ITR3: Internal trigger 3 selected
  - TIM_TS_ITR4: Internal trigger 4 selected
  - TIM_TS_ITR5: Internal trigger 5 selected
  - TIM_TS_ITR6: Internal trigger 6 selected
  - TIM_TS_ITR7: Internal trigger 7 selected
  - TIM_TS_ITR8: Internal trigger 8 selected
  - TIM_TS_ITR11: Internal trigger 11 selected
  - TIM_TS_NONE: No trigger is needed
- **CommutationSource:** the Commutation Event source This parameter can be one of the following values:
  - TIM_COMMUTATION_TRGI: Commutation source is the TRGI of the Interface Timer
  - TIM_COMMUTATION_SOFTWARE: Commutation source is set by software using the COMG bit

Return values

- **HAL:** status

Notes

- This function is mandatory to use the commutation event in order to update the configuration at each commutation detection on the TRGI input of the Timer, the typical use of this feature is with the use of another Timer(interface Timer) configured in Hall sensor interface, this interface Timer will generate the commutation at its TRGO output (connected to Timer used in this function) each time the TI1 of the Interface Timer detect a commutation at its input TI1.
- The user should configure the DMA in his own software, in This function only the COMDE bit is set

HAL_TIMEx_MasterConfigSynchronization

Function name

HAL_StatusTypeDef HAL_TIMEx_MasterConfigSynchronization (TIM_HandleTypeDef * htim, const TIM_MasterConfigTypeDef * sMasterConfig)

Function description

Configures the TIM in master mode.

Parameters

- **htim:** TIM handle.
- **sMasterConfig:** pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.

Return values

- **HAL:** status

HAL_TIMEx_ConfigBreakDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakDeadTime (TIM_HandleTypeDef * htim, const TIM_BreakDeadTimeConfigTypeDef * sBreakDeadTimeConfig)

Function description

Configures the Break feature, dead time, Lock level, OSSI/OSSR State and the AOE(automatic output enable).

Parameters

- **htim:** TIM handle
- **sBreakDeadTimeConfig:** pointer to a TIM_ConfigBreakDeadConfigTypeDef structure that contains the BDTR Register configuration information for the TIM peripheral.

Return values

- **HAL:** status

Notes

- Interrupts can be generated when an active level is detected on the break input, the break 2 input or the system break input. Break interrupt can be enabled by calling the __HAL_TIM_ENABLE_IT macro.

HAL_TIMEx_ConfigBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput, const TIMEx_BreakInputConfigTypeDef * sBreakInputConfig)

Function description

Configures the break input source.

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to configure This parameter can be one of the following values:
  – TIM_BREAKINPUT_BRK: Timer break input
  – TIM_BREAKINPUT_BRK2: Timer break 2 input
- **sBreakInputConfig:** Break input source configuration

Return values

- **HAL:** status

HAL_TIMEx_GroupChannel5

Function name

HAL_StatusTypeDef HAL_TIMEx_GroupChannel5 (TIM_HandleTypeDef * htim, uint32_t Channels)

Function description

Group channel 5 and channel 1, 2 or 3.

Parameters

- **htim:** TIM handle.
- **Channels:** specifies the reference signal(s) the OC5REF is combined with. This parameter can be any combination of the following values: TIM_GROUPCH5_NONE: No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC TIM_GROUPCH5_OC1REFC: OC1REFC is the logical AND of OC1REFC and OC5REF TIM_GROUPCH5_OC2REFC: OC2REFC is the logical AND of OC2REFC and OC5REF TIM_GROUPCH5_OC3REFC: OC3REFC is the logical AND of OC3REFC and OC5REF

Return values

- **HAL:** status

HAL_TIMEx_RemapConfig

Function name

HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)

Function description

Configures the TIMx Remapping input capabilities.

Parameters

- **htim:** TIM handle.
- **Remap:** specifies the TIM remapping source. For TIM1, the parameter can take one of the following values:
  - TIM_TIM1_ETR_GPIO TIM1 ETR is connected to GPIO
  - TIM_TIM1_ETR_COMP1 TIM1 ETR is connected to COMP1 output
  - TIM_TIM1_ETR_COMP2 TIM1 ETR is connected to COMP2 output (*)
  - TIM_TIM1_ETR_HSI TIM1 ETR is connected to HSI
  - TIM_TIM1_ETR_MSIS TIM1_ETR is connected to MSIS
  - TIM_TIM1_ETR_ADC2_AWD2 TIM1_ETR is connected to ADC2 AWD2 (*)
  - TIM_TIM1_ETR_ADC2_AWD3 TIM1_ETR is connected to ADC2 AWD3 (*)
  - TIM_TIM1_ETR_ADC1_AWD1 TIM1 ETR is connected to ADC1 AWD1
  - TIM_TIM1_ETR_ADC1_AWD2 TIM1 ETR is connected to ADC1 AWD2
  - TIM_TIM1_ETR_ADC1_AWD3 TIM1 ETR is connected to ADC1 AWD3
  - TIM_TIM1_ETR_ADC4_AWD1 TIM1 ETR is connected to ADC4 AWD1
  - TIM_TIM1_ETR_ADC4_AWD2 TIM1 ETR is connected to ADC4 AWD2
  - TIM_TIM1_ETR_ADC4_AWD3 TIM1 ETR is connected to ADC4 AWD3
  - TIM_TIM1_ETR_ADC2_AWD1 TIM1_ETR is connected to ADC2 AWD1 (*)

  For TIM2, the parameter can take one of the following values:
  - TIM_TIM2_ETR_GPIO TIM2 ETR is connected to GPIO
  - TIM_TIM2_ETR_COMP1 TIM2 ETR is connected to COMP1 output
  - TIM_TIM2_ETR_COMP2 TIM2 ETR is connected to COMP2 output (*)
  - TIM_TIM2_ETR_MSIK TIM2 ETR is connected to MSIK
  - TIM_TIM2_ETR_HSI TIM2 ETR is connected to HSI
  - TIM_TIM2_ETR_MSIS TIM2_ETR is connected to MSIS
  - TIM_TIM2_ETR_DCMI_VSYNC TIM2_ETR is connected to DCMI VSYNC (*)
  - TIM_TIM2_ETR_LTDC_VSYNC TIM2_ETR is connected to LTDC_VSYNC (*)
  - TIM_TIM2_ETR_TIM3_ETR TIM2 ETR is connected to TIM3 ETR pin
  - TIM_TIM2_ETR_TIM4_ETR TIM2 ETR is connected to TIM4 ETR pin
  - TIM_TIM2_ETR_TIM5_ETR TIM2 ETR is connected to TIM5 ETR pin
  - TIM_TIM2_ETR_LSE TIM2 ETR is connected to LSE
  - TIM_TIM2_ETR_DSI_TE TIM2_ETR is connected to DSI_TE (*)
  - TIM_TIM2_ETR_DCMI_HSYNC TIM2_ETR is connected to DCMI HSYNC (*)
  - TIM_TIM2_ETR_LTDC_HSYNC TIM2_ETR is connected to LTDC HSYNC (*)

  For TIM3, the parameter can take one of the following values:
  - TIM_TIM3_ETR_GPIO TIM3 ETR is connected to GPIO
  - TIM_TIM3_ETR_COMP1 TIM3 ETR is connected to COMP1 output
  - TIM_TIM3_ETR_COMP2 TIM3 ETR is connected to COMP2 output (*)
  - TIM_TIM3_ETR_MSIK TIM3 ETR is connected to MSIK
  - TIM_TIM3_ETR_HSI TIM3 ETR is connected to HSI
  - TIM_TIM3_ETR_MSIS TIM3_ETR is connected to MSIS
  - TIM_TIM3_ETR_DCMI_VSYNC TIM3_ETR is connected to DCMI VSYNC (*)
  - TIM_TIM3_ETR_LTDC_VSYNC TIM3_ETR is connected to LTDC_VSYNC (*)
  - TIM_TIM3_ETR_TIM2_ETR TIM3 ETR is connected to TIM2 ETR pin
  - TIM_TIM3_ETR_TIM4_ETR TIM3 ETR is connected to TIM4 ETR pin
  - TIM_TIM3_ETR_DSI_TE TIM2_ETR is connected to DSI_TE (*)
  - TIM_TIM3_ETR_ADC1_AWD1 TIM3 ETR is connected to ADC1 AWD1
  - TIM_TIM3_ETR_ADC1_AWD2 TIM3 ETR is connected to ADC1 AWD2
  - TIM_TIM3_ETR_ADC1_AWD3 TIM3 ETR is connected to ADC1 AWD3
  - TIM_TIM3_ETR_DCMI_HSYNC TIM3_ETR is connected to DCMI HSYNC (*)

– TIM_TIM3_ETR_LTDC_HSYNC TIM3_ETR is connected to LTDC HSYNC (*)

For TIM4, the parameter can take one of the following values:

– TIM_TIM4_ETR_GPIO TIM4 ETR is connected to GPIO
– TIM_TIM4_ETR_COMP1 TIM4 ETR is connected to COMP1 output
– TIM_TIM4_ETR_COMP2 TIM4 ETR is connected to COMP2 output (*)
– TIM_TIM4_ETR_MSIK TIM4 ETR is connected to MSIK
– TIM_TIM4_ETR_HSI TIM4 ETR is connected to HSI
– TIM_TIM4_ETR_MSIS TIM4_ETR is connected to MSIS
– TIM_TIM4_ETR_DCMI_VSYNC TIM4_ETR is connected to DCMI VSYNC (*)
– TIM_TIM4_ETR_LTDC_VSYNC TIM4_ETR is connected to LTDC_VSYNC (*)
– TIM_TIM4_ETR_TIM3_ETR TIM4 ETR is connected to TIM3 ETR pin
– TIM_TIM4_ETR_TIM5_ETR TIM4 ETR is connected to TIM5 ETR pin
– TIM_TIM4_ETR_DSI_TE TIM2_ETR is connected to DSI_TE (*)
– TIM_TIM4_ETR_ADC2_AWD1 TIM4_ETR is connected to ADC2 AWD1 (*)
– TIM_TIM4_ETR_ADC2_AWD2 TIM4_ETR is connected to ADC2 AWD2 (*)
– TIM_TIM4_ETR_ADC2_AWD3 TIM4_ETR is connected to ADC2 AWD3 (*)
– TIM_TIM4_ETR_DCMI_HSYNC TIM4_ETR is connected to DCMI HSYNC (*)
– TIM_TIM4_ETR_LTDC_HSYNC TIM4_ETR is connected to LTDC HSYNC (*)

For TIM5, the parameter can take one of the following values:

– TIM_TIM5_ETR_GPIO TIM5 ETR is connected to GPIO
– TIM_TIM5_ETR_COMP1 TIM5 ETR is connected to COMP1 output
– TIM_TIM5_ETR_COMP2 TIM5 ETR is connected to COMP2 output (*)
– TIM_TIM5_ETR_MSIK TIM5 ETR is connected to MSIK
– TIM_TIM5_ETR_HSI TIM5 ETR is connected to HSI
– TIM_TIM5_ETR_MSIS TIM5_ETR is connected to MSIS
– TIM_TIM5_ETR_DCMI_VSYNC TIM5_ETR is connected to DCMI VSYNC (*)
– TIM_TIM5_ETR_LTDC_VSYNC TIM5_ETR is connected to LTDC_VSYNC (*)
– TIM_TIM5_ETR_TIM2_ETR TIM5 ETR is connected to TIM2 ETR pin
– TIM_TIM5_ETR_TIM3_ETR TIM5 ETR is connected to TIM3 ETR pin
– TIM_TIM5_ETR_DSI_TE TIM5_ETR is connected to DSI_TE (*)
– TIM_TIM5_ETR_DCMI_HSYNC TIM5_ETR is connected to DCMI HSYNC (*)
– TIM_TIM5_ETR_LTDC_HSYNC TIM5_ETR is connected to LTDC HSYNC (*)

For TIM8, the parameter can take one of the following values:

– TIM_TIM8_ETR_GPIO TIM8 ETR is connected to GPIO
– TIM_TIM8_ETR_COMP1 TIM8 ETR is connected to COMP1 output
– TIM_TIM8_ETR_COMP2 TIM8 ETR is connected to COMP2 output (*)
– TIM_TIM8_ETR_MSIK TIM8 ETR is connected to MSIK
– TIM_TIM8_ETR_HSI TIM8 ETR is connected to HSI
– TIM_TIM8_ETR_MSIS TIM8_ETR is connected to MSIS
– TIM_TIM8_ETR_ADC2_AWD2 TIM8_ETR is connected to ADC2 AWD2 (*)
– TIM_TIM8_ETR_ADC2_AWD3 TIM8_ETR is connected to ADC2 AWD3 (*)
– TIM_TIM8_ETR_ADC1_AWD1 TIM8 ETR is connected to ADC1 AWD1
– TIM_TIM8_ETR_ADC1_AWD2 TIM8 ETR is connected to ADC1 AWD2
– TIM_TIM8_ETR_ADC1_AWD3 TIM8 ETR is connected to ADC1 AWD3
– TIM_TIM8_ETR_ADC4_AWD1 TIM8 ETR is connected to ADC4 AWD1
– TIM_TIM8_ETR_ADC4_AWD2 TIM8 ETR is connected to ADC4 AWD2
– TIM_TIM8_ETR_ADC4_AWD3 TIM8 ETR is connected to ADC4 AWD3
– TIM_TIM8_ETR_ADC2_AWD1 TIM8_ETR is connected to ADC2 AWD1 (*)

(*) Value not defined in all devices.

Return values

- **HAL:** status

HAL_TIMEx_TISelection

Function name

HAL_StatusTypeDef HAL_TIMEx_TISelection (TIM_HandleTypeDef * htim, uint32_t TISelection, uint32_t Channel)

Function description

Select the timer input source.

Parameters

- **htim:** TIM handle.
- **Channel:** specifies the TIM Channel This parameter can be one of the following values:
  – TIM_CHANNEL_1: TI1 input channel
  – TIM_CHANNEL_2: TI2 input channel
  – TIM_CHANNEL_4: TIM Channel 4

- **TISelection:** parameter of the TIM_TISelectionStruct structure is detailed as follows: For TIM1, the parameter is one of the following values:
  – TIM_TIM1_TI1_GPIO: TIM1 TI1 is connected to GPIO
  – TIM_TIM1_TI1_COMP1: TIM1 TI1 is connected to COMP1 output
  – TIM_TIM1_TI1_COMP2: TIM1 TI1 is connected to COMP2 output (*)

  For TIM2, the parameter is one of the following values:
  – TIM_TIM2_TI1_GPIO: TIM2 TI1 is connected to GPIO
  – TIM_TIM2_TI1_COMP1: TIM2 TI1 is connected to COMP1 output
  – TIM_TIM2_TI1_COMP2: TIM2 TI1 is connected to COMP2 output (*)
  – TIM_TIM2_TI2_GPIO: TIM2 TI2 is connected to GPIO
  – TIM_TIM2_TI2_COMP1: TIM2 TI2 is connected to COMP1 output
  – TIM_TIM2_TI2_COMP2: TIM2 TI2 is connected to COMP2 output (*)
  – TIM_TIM2_TI4_GPIO: TIM2 TI4 is connected to GPIO
  – TIM_TIM2_TI4_COMP1: TIM2 TI4 is connected to COMP1 output
  – TIM_TIM2_TI4_COMP2: TIM2 TI4 is connected to COMP2 output (*)

  For TIM3, the parameter is one of the following values:
  – TIM_TIM3_TI1_GPIO: TIM3 TI1 is connected to GPIO
  – TIM_TIM3_TI1_COMP1: TIM3 TI1 is connected to COMP1 output
  – TIM_TIM3_TI1_COMP2: TIM3 TI1 is connected to COMP2 output (*)
  – TIM_TIM3_TI2_GPIO: TIM3 TI2 is connected to GPIO
  – TIM_TIM3_TI2_COMP1: TIM3 TI2 is connected to COMP1 output
  – TIM_TIM3_TI2_COMP2: TIM3 TI2 is connected to COMP2 output (*)

  For TIM4, the parameter is one of the following values:
  – TIM_TIM4_TI1_GPIO: TIM4 TI1 is connected to GPIO
  – TIM_TIM4_TI1_COMP1: TIM4 TI1 is connected to COMP1 output
  – TIM_TIM4_TI1_COMP2: TIM4 TI1 is connected to COMP2 output (*)
  – TIM_TIM4_TI2_GPIO: TIM4 TI2 is connected to GPIO
  – TIM_TIM4_TI2_COMP1: TIM4 TI2 is connected to COMP1 output
  – TIM_TIM4_TI2_COMP2: TIM4 TI2 is connected to COMP2 output (*)

  For TIM5, the parameter is one of the following values:
  – TIM_TIM5_TI1_GPIO: TIM5 TI1 is connected to GPIO
  – TIM_TIM5_TI1_LSI: TIM5 TI1 is connected to LSI
  – TIM_TIM5_TI1_LSE: TIM5 TI1 is connected to LSE
  – TIM_TIM5_TI1_RTC_WKUP: TIM5 TI1 is connected to RTC wakeup interrupt
  – TIM_TIM5_TI1_COMP1: TIM5 TI1 is connected to COMP1 output
  – TIM_TIM5_TI1_COMP2: TIM5 TI1 is connected to COMP2 output (*)
  – TIM_TIM5_TI2_GPIO: TIM5 TI2 is connected to GPIO
  – TIM_TIM5_TI2_COMP1: TIM5 TI2 is connected to COMP1 output
  – TIM_TIM5_TI2_COMP2: TIM5 TI2 is connected to COMP2 output (*)

  For TIM8, the parameter is one of the following values:
  – TIM_TIM8_TI1_GPIO: TIM8 TI1 is connected to GPIO
  – TIM_TIM8_TI1_COMP1: TIM8 TI1 is connected to COMP1 output
  – TIM_TIM8_TI1_COMP2: TIM8 TI1 is connected to COMP2 output (*)

  For TIM15, the parameter is one of the following values:
  – TIM_TIM15_TI1_GPIO: TIM15 TI1 is connected to GPIO
  – TIM_TIM15_TI1_LSE: TIM15 TI1 is connected to LSE
  – TIM_TIM15_TI1_COMP1: TIM15 TI1 is connected to COMP1 output
  – TIM_TIM15_TI1_COMP2: TIM15 TI1 is connected to COMP2 output (*)
  – TIM_TIM15_TI2_GPIO: TIM15 TI2 is connected to GPIO
  – TIM_TIM15_TI2_TIM2: TIM15 TI2 is connected to TIM2 CH2

- TIM_TIM15_TI2_TIM3: TIM15 TI2 is connected to TIM3 CH2
- TIM_TIM15_TI2_COMP2: TIM15 TI2 is connected to COMP2 output (*)

For TIM16, the parameter can have the following values:

- TIM_TIM16_TI1_GPIO: TIM16 TI1 is connected to GPIO
- TIM_TIM16_TI1_MCO: TIM16 TI1 is connected to MCO
- TIM_TIM16_TI1_LSI: TIM16 TI1 is connected to LSI
- TIM_TIM16_TI1_LSE: TIM16 TI1 is connected to LSE
- TIM_TIM16_TI1_RTC_WKUP: TIM16 TI1 is connected to RTC wakeup interrupt
- TIM_TIM16_TI1_HSE_DIV32: TIM16 TI1 is connected to HSE/32
- TIM_TIM16_TI1_MSIS_1024: TIM16 TI1 is connected to MSIS/1024
- TIM_TIM16_TI1_MSIS_4: TIM16 TI1 is connected to MSIS/4
- TIM_TIM16_TI1_HSI_256: TIM16 TI1 is connected to HSI/256

For TIM17, the parameter can have the following values:

- TIM_TIM17_TI1_GPIO: TIM17 TI1 is connected to GPIO
- TIM_TIM17_TI1_MCO: TIM17 TI1 is connected to MCO
- TIM_TIM17_TI1_LSI: TIM17 TI1 is connected to LSI
- TIM_TIM17_TI1_LSE: TIM17 TI1 is connected to LSE
- TIM_TIM17_TI1_RTC_WKUP: TIM17 TI1 is connected to RTC wakeup interrupt
- TIM_TIM17_TI1_HSE_DIV32: TIM17 TI1 is connected to HSE/32
- TIM_TIM17_TI1_MSIS_1024: TIM17 TI1 is connected to MSIS/024
- TIM_TIM17_TI1_MSIS_4: TIM17 TI1 is connected to MSIS/4
- TIM_TIM17_TI1_HSI_256: TIM17 TI1 is connected to HSI/256

(*) Value not defined in all devices.

•

Return values

- **HAL:** status

HAL_TIMEx_EnableHSE32

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableHSE32 (TIM_HandleTypeDef * htim)

Function description

Enable HSE/32 .

Parameters

- **htim:** TIM handle.

Return values

- **HAL:** status.

Notes

- This function should be called to enable the HSE/32 when it is selected as TIM17_TI1 or TIM16_TI1 input source.

HAL_TIMEx_DisableHSE32

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableHSE32 (TIM_HandleTypeDef * htim)

Function description

Disable HSE/32 .

Parameters

- **htim:** TIM handle.

Return values

- **HAL:** status.

Notes

- This function should be called to Disable the HSE/32 .

HAL_TIMEx_DisarmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_DisarmBreakInput (TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

Disarm the designated break input (when it operates in bidirectional mode).

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to disarm This parameter can be one of the following values:
  – TIM_BREAKINPUT_BRK: Timer break input
  – TIM_BREAKINPUT_BRK2: Timer break 2 input

Return values

- **HAL:** status

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

HAL_TIMEx_ReArmBreakInput

Function name

HAL_StatusTypeDef HAL_TIMEx_ReArmBreakInput (const TIM_HandleTypeDef * htim, uint32_t BreakInput)

Function description

Arm the designated break input (when it operates in bidirectional mode).

Parameters

- **htim:** TIM handle.
- **BreakInput:** Break input to arm This parameter can be one of the following values:
  – TIM_BREAKINPUT_BRK: Timer break input
  – TIM_BREAKINPUT_BRK2: Timer break 2 input

Return values

- **HAL:** status

Notes

- Arming is possible at anytime, even if fault is present.
- Break input is automatically armed as soon as MOE bit is set.

HAL_TIMEx_DitheringEnable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringEnable (TIM_HandleTypeDef * htim)

Function description

Enable dithering.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

Notes

- Main usage is PWM mode
- This function must be called when timer is stopped or disabled (CEN =0)
- If dithering is activated, pay attention to ARR, CCRx, CNT interpretation: CNT: only CNT[11:0] holds the non-dithered part for 16b timers (or CNT[26:0] for 32b timers)ARR: ARR[15:4] holds the non-dithered part, and ARR[3:0] the dither part for 16b timersCCRx: CCRx[15:4] holds the non-dithered part, and CCRx[3:0] the dither part for 16b timersARR and CCRx values are limited to 0xFFEF in dithering mode for 16b timers (corresponds to 4094 for the integer part and 15 for the dithered part).
- Macros __HAL_TIM_CALC_PERIOD_DITHER() __HAL_TIM_CALC_DELAY_DITHER() __HAL_TIM_CALC_PULSE_DITHER() can be used to calculate period (ARR) and delay (CCRx) value.
- Enabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part.
- Enabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part. So it may be necessary to read ARR value or CCRx value with macros __HAL_TIM_GET_AUTORELOAD() __HAL_TIM_GET_COMPARE() and if necessary update Init structure field htim->Init.Period .

HAL_TIMEx_DitheringDisable

Function name

HAL_StatusTypeDef HAL_TIMEx_DitheringDisable (TIM_HandleTypeDef * htim)

Function description

Disable dithering.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

Notes

- This function must be called when timer is stopped or disabled (CEN =0)
- If dithering is activated, pay attention to ARR, CCRx, CNT interpretation: CNT: only CNT[11:0] holds the non-dithered part for 16b timers (or CNT[26:0] for 32b timers)ARR: ARR[15:4] holds the non-dithered part, and ARR[3:0] the dither part for 16b timersCCRx: CCRx[15:4] holds the non-dithered part, and CCRx[3:0] the dither part for 16b timersARR and CCRx values are limited to 0xFFEF in dithering mode (corresponds to 4094 for the integer part and 15 for the dithered part).
- Disabling dithering, modifies automatically values of registers ARR/CCRx to keep the same integer part. So it may be necessary to read ARR value or CCRx value with macros __HAL_TIM_GET_AUTORELOAD() __HAL_TIM_GET_COMPARE() and if necessary update Init structure field htim->Init.Period .

HAL_TIMEx_OC_ConfigPulseOnCompare

Function name

HAL_StatusTypeDef HAL_TIMEx_OC_ConfigPulseOnCompare (TIM_HandleTypeDef * htim, uint32_t PulseWidthPrescaler, uint32_t PulseWidth)

Function description

Initializes the pulse on compare pulse width and pulse prescaler.

Parameters

- **htim:** TIM Output Compare handle
- **PulseWidthPrescaler:** Pulse width prescaler This parameter can be a number between Min_Data = 0x0 and Max_Data = 0x7
- **PulseWidth:** Pulse width This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

Return values

- **HAL:** status

HAL_TIMEx_ConfigSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigSlaveModePreload (TIM_HandleTypeDef * htim, uint32_t Source)

Function description

Configure preload source of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle
- **Source:** Source of slave mode selection preload This parameter can be one of the following values:
  – TIM_SMS_PRELOAD_SOURCE_UPDATE: Timer update event is used as source of Slave Mode Selection preload
  – TIM_SMS_PRELOAD_SOURCE_INDEX: Timer index event is used as source of Slave Mode Selection preload

Return values

- **HAL:** status

HAL_TIMEx_EnableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

Enable preload of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableSlaveModePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableSlaveModePreload (TIM_HandleTypeDef * htim)

Function description

Disable preload of Slave Mode Selection bitfield (SMS in SMCR register)

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_EnableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

Enable deadtime preload.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableDeadTimePreload

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableDeadTimePreload (TIM_HandleTypeDef * htim)

Function description

Disable deadtime preload.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_ConfigDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigDeadTime (TIM_HandleTypeDef * htim, uint32_t Deadtime)

Function description

Configure deadtime.

Parameters

- **htim:** TIM handle
- **Deadtime:** Deadtime value

Return values

- **HAL:** status

Notes

- This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

HAL_TIMEx_ConfigAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigAsymmetricalDeadTime (TIM_HandleTypeDef * htim, uint32_t FallingDeadtime)

Function description

Configure asymmetrical deadtime.

Parameters

- **htim:** TIM handle
- **FallingDeadtime:** Falling edge deadtime value

Return values

- **HAL:** status

Notes

- This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF

HAL_TIMEx_EnableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

Enable asymmetrical deadtime.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableAsymmetricalDeadTime

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableAsymmetricalDeadTime (TIM_HandleTypeDef * htim)

Function description

Disable asymmetrical deadtime.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_ConfigEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_ConfigEncoderIndex (TIM_HandleTypeDef * htim, TIMEx_EncoderIndexConfigTypeDef * sEncoderIndexConfig)

Function description

Configures the encoder index.

Parameters

- **htim:** TIM handle.
- **sEncoderIndexConfig:** Encoder index configuration

Return values

- **HAL:** status

Notes

- warning in case of encoder mode clock plus direction TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1 or TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2 Direction must be set to TIM_ENCODERINDEX_DIRECTION_UP_DOWN

HAL_TIMEx_EnableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

Enable encoder index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableEncoderIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderIndex (TIM_HandleTypeDef * htim)

Function description

Disable encoder index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_EnableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_EnableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

Enable encoder first index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_DisableEncoderFirstIndex

Function name

HAL_StatusTypeDef HAL_TIMEx_DisableEncoderFirstIndex (TIM_HandleTypeDef * htim)

Function description

Disable encoder first index.

Parameters

- **htim:** TIM handle

Return values

- **HAL:** status

HAL_TIMEx_CommutCallback

Function name

void HAL_TIMEx_CommutCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_CommutHalfCpltCallback

Function name

void HAL_TIMEx_CommutHalfCpltCallback (TIM_HandleTypeDef * htim)

Function description

Hall commutation changed half complete callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_BreakCallback

Function name

void HAL_TIMEx_BreakCallback (TIM_HandleTypeDef * htim)

Function description

Hall Break detection callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_Break2Callback

Function name

void HAL_TIMEx_Break2Callback (TIM_HandleTypeDef * htim)

Function description

Hall Break2 detection callback in non blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_EncoderIndexCallback

Function name

void HAL_TIMEx_EncoderIndexCallback (TIM_HandleTypeDef * htim)

Function description

Encoder index callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_DirectionChangeCallback

Function name

void HAL_TIMEx_DirectionChangeCallback (TIM_HandleTypeDef * htim)

Function description

Direction change callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_IndexErrorCallback

Function name

void HAL_TIMEx_IndexErrorCallback (TIM_HandleTypeDef * htim)

Function description

Index error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_TransitionErrorCallback

Function name

void HAL_TIMEx_TransitionErrorCallback (TIM_HandleTypeDef * htim)

Function description

Transition error callback in non-blocking mode.

Parameters

- **htim:** TIM handle

Return values

- **None:**

HAL_TIMEx_HallSensor_GetState

Function name

HAL_TIM_StateTypeDef HAL_TIMEx_HallSensor_GetState (const TIM_HandleTypeDef * htim)

Function description

Return the TIM Hall Sensor interface handle state.

Parameters

- **htim:** TIM Hall Sensor handle

Return values

- **HAL:** state

HAL_TIMEx_GetChannelNState

Function name

HAL_TIM_ChannelStateTypeDef HAL_TIMEx_GetChannelNState (const TIM_HandleTypeDef * htim, uint32_t ChannelN)

Function description

Return actual state of the TIM complementary channel.

Parameters

- **htim:** TIM handle
- **ChannelN:** TIM Complementary channel This parameter can be one of the following values:
  – TIM_CHANNEL_1: TIM Channel 1
  – TIM_CHANNEL_2: TIM Channel 2
  – TIM_CHANNEL_3: TIM Channel 3
  – TIM_CHANNEL_4: TIM Channel 4

Return values

- **TIM:** Complementary channel state

TIMEx_DMACommutationCplt

Function name

void TIMEx_DMACommutationCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

TIMEx_DMACommutationHalfCplt

Function name

void TIMEx_DMACommutationHalfCplt (DMA_HandleTypeDef * hdma)

Function description

TIM DMA Commutation half complete callback.

Parameters

- **hdma:** pointer to DMA handle.

Return values

- **None:**

82.3 TIMEx Firmware driver defines

The following section lists the various define and macros of the module.

82.3.1 TIMEx

TIMEx

TIM Extended Break input

TIM_BREAKINPUT_BRK

Timer break input

TIM_BREAKINPUT_BRK2

Timer break2 input

TIM Extended Break input source

TIM_BREAKINPUTSOURCE_BKIN

An external source (GPIO) is connected to the BKIN pin

TIM_BREAKINPUTSOURCE_COMP1

The COMP1 output is connected to the break input

TIM_BREAKINPUTSOURCE_COMP2

The COMP2 output is connected to the break input

TIM_BREAKINPUTSOURCE_MDF1

The analog watchdog output of the MDF1 peripheral is connected to the break input

TIM Extended Break input source enabling

TIM_BREAKINPUTSOURCE_DISABLE

Break input source is disabled

TIM_BREAKINPUTSOURCE_ENABLE

Break input source is enabled

TIM Extended Break input polarity

TIM_BREAKINPUTSOURCE_POLARITY_LOW

Break input source is active low

TIM_BREAKINPUTSOURCE_POLARITY_HIGH

Break input source is active_high

TIM Extended Encoder index blanking

TIM_ENCODERINDEX_BLANKING_DISABLE

Encoder index blanking is disabled

TIM_ENCODERINDEX_BLANKING_TI3

Encoder index blanking is enabled on TI3

TIM_ENCODERINDEX_BLANKING_TI4

Encoder index blanking is enabled on TI4

TIM Extended Encoder index direction

TIM_ENCODERINDEX_DIRECTION_UP_DOWN

Index resets the counter whatever the direction

TIM_ENCODERINDEX_DIRECTION_UP

Index resets the counter when up-counting only

TIM_ENCODERINDEX_DIRECTION_DOWN

Index resets the counter when down-counting only

TIM Extended Encoder index polarity

TIM_ENCODERINDEX_POLARITY_INVERTED

Polarity for ETRx pin

TIM_ENCODERINDEX_POLARITY_NONINVERTED

Polarity for ETRx pin

TIM Extended Encoder index position

TIM_ENCODERINDEX_POSITION_00

Encoder index position is AB=00

TIM_ENCODERINDEX_POSITION_01

Encoder index position is AB=01

TIM_ENCODERINDEX_POSITION_10

Encoder index position is AB=10

TIM_ENCODERINDEX_POSITION_11

Encoder index position is AB=11

TIM_ENCODERINDEX_POSITION_0

In directional clock mode or clock plus direction mode, index resets the counter when clock is 0

TIM_ENCODERINDEX_POSITION_1

In directional clock mode or clock plus direction mode, index resets the counter when clock is 1

TIM Extended Encodder index prescaler

TIM_ENCODERINDEX_PRESCALER_DIV1

No prescaler is used

TIM_ENCODERINDEX_PRESCALER_DIV2

Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_ENCODERINDEX_PRESCALER_DIV4

Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_ENCODERINDEX_PRESCALER_DIV8

Prescaler for External ETR pin: Capture performed once every 8 events.

TIM Extended Exported Macros

__HAL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PSC(80000000, 1000000);

__HAL_TIM_CALC_PERIOD

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __HAL_TIM_CALC_PERIOD(1000000, 0, 10000);

__HAL_TIM_CALC_PERIOD_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER(1000000, 0, 10000); This macro should be used only if dithering is already enabled

__HAL_TIM_CALC_PULSE

Description:
- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:
- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:
- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:
- ex: __HAL_TIM_CALC_PULSE(1000000, 0, 10);

__HAL_TIM_CALC_PULSE_DITHER

Description:
- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:
- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:
- Compare: value (between Min_Data=0 and Max_Data=65519)

Notes:
- ex: __HAL_TIM_CALC_PULSE_DITHER(1000000, 0, 10); This macro should be used only if dithering is already enabled

__HAL_TIM_CALC_PERIOD_BY_DELAY

Description:
- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:
- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:
- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:
- ex: __HAL_TIM_CALC_PERIOD_BY_DELAY(1000000, 0, 10, 20);

__HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65519)

Notes:

- ex: __HAL_TIM_CALC_PERIOD_DITHER_BY_DELAY(1000000, 0, 10, 20); This macro should be used only if dithering is already enabled

TIM Extended Remapping

TIM_TIM1_ETR_GPIO

TIM1_ETR is not connected to I/O

TIM_TIM1_ETR_COMP1

TIM1_ETR is connected to COMP1 output

TIM_TIM1_ETR_COMP2

TIM1_ETR is connected to COMP2 output

TIM_TIM1_ETR_MSIK

TIM1_ETR is connected to MSIK

TIM_TIM1_ETR_HSI

TIM1_ETR is connected to HSI

TIM_TIM1_ETR_MSIS

TIM1_ETR is connected to MSI

TIM_TIM1_ETR_ADC2_AWD2

TIM1_ETR is connected to ADC2 AWD2

TIM_TIM1_ETR_ADC2_AWD3

TIM1_ETR is connected to ADC2 AWD3

TIM_TIM1_ETR_ADC1_AWD1

TIM1_ETR is connected to ADC1 AWD1

TIM_TIM1_ETR_ADC1_AWD2

TIM1_ETR is connected to ADC1 AWD2

TIM_TIM1_ETR_ADC1_AWD3

TIM1_ETR is connected to ADC1 AWD3

TIM_TIM1_ETR_ADC4_AWD1

TIM1_ETR is connected to ADC4 AWD1

TIM_TIM1_ETR_ADC4_AWD2

TIM1_ETR is connected to ADC4 AWD2

TIM_TIM1_ETR_ADC4_AWD3
TIM1_ETR is connected to ADC4 AWD3

TIM_TIM1_ETR_ADC2_AWD1
TIM1_ETR is connected to ADC2 AWD1

TIM_TIM2_ETR_GPIO
TIM2_ETR is not connected to I/O

TIM_TIM2_ETR_COMP1
TIM2_ETR is connected to COMP1 output

TIM_TIM2_ETR_COMP2
TIM2_ETR is connected to COMP2 output

TIM_TIM2_ETR_MSIK
TIM2_ETR is connected to MSIK

TIM_TIM2_ETR_HSI
TIM2_ETR is connected to HSI

TIM_TIM2_ETR_MSIS
TIM2_ETR is connected to MSIS

TIM_TIM2_ETR_DCMI_VSYNC
TIM2_ETR is connected to DCMI VSYNC

TIM_TIM2_ETR_LTDC_VSYNC
TIM2_ETR is connected to LTDC_VSYNC

TIM_TIM2_ETR_TIM3_ETR
TIM2_ETR is connected to TIM3 ETR

TIM_TIM2_ETR_TIM4_ETR
TIM2_ETR is connected to TIM4 ETR

TIM_TIM2_ETR_TIM5_ETR
TIM2_ETR is connected to TIM5 ETR

TIM_TIM2_ETR_LSE
TIM2_ETR is connected to LSE

TIM_TIM2_ETR_DSI_TE
TIM2_ETR is connected to DSI_TE

TIM_TIM2_ETR_DCMI_HSYNC
TIM2_ETR is connected to DCMI HSYNC

TIM_TIM2_ETR_LTDC_HSYNC
TIM2_ETR is connected to LTDC HSYNC

TIM_TIM3_ETR_GPIO
TIM3_ETR is not connected to I/O

TIM_TIM3_ETR_COMP1
TIM3_ETR is connected to COMP1 output

TIM_TIM3_ETR_COMP2

TIM3_ETR is connected to COMP2 output

TIM_TIM3_ETR_MSIK

TIM3_ETR is connected to MSIK

TIM_TIM3_ETR_HSI

TIM3_ETR is connected to HSI

TIM_TIM3_ETR_MSIS

TIM3_ETR is connected to MSIS

TIM_TIM3_ETR_DCMI_VSYNC

TIM3_ETR is connected to DCMI VSYNC

TIM_TIM3_ETR_LTDC_VSYNC

TIM3_ETR is connected to LTDC_VSYNC

TIM_TIM3_ETR_TIM2_ETR

TIM3_ETR is connected to TIM2 ETR

TIM_TIM3_ETR_TIM4_ETR

TIM3_ETR is connected to TIM4 ETR

TIM_TIM3_ETR_DSI_TE

TIM3_ETR is connected to DSI_TE

TIM_TIM3_ETR_ADC1_AWD1

TIM3_ETR is connected to ADC1 AWD1

TIM_TIM3_ETR_ADC1_AWD2

TIM3_ETR is connected to ADC1 AWD2

TIM_TIM3_ETR_ADC1_AWD3

TIM3_ETR is connected to ADC1 AWD3

TIM_TIM3_ETR_DCMI_HSYNC

TIM3_ETR is connected to DCMI HSYNC

TIM_TIM3_ETR_LTDC_HSYNC

TIM3_ETR is connected to LTDC HSYNC

TIM_TIM4_ETR_GPIO

TIM4_ETR is not connected to I/O

TIM_TIM4_ETR_COMP1

TIM4_ETR is connected to COMP1 output

TIM_TIM4_ETR_COMP2

TIM4_ETR is connected to COMP2 output

TIM_TIM4_ETR_MSIK

TIM4_ETR is connected to MSIK

TIM_TIM4_ETR_HSI

TIM4_ETR is connected to HSI

TIM_TIM4_ETR_MSIS
TIM4_ETR is connected to MSIS

TIM_TIM4_ETR_DCMI_VSYNC
TIM4_ETR is connected to DCMI VSYNC

TIM_TIM4_ETR_LTDC_VSYNC
TIM4_ETR is connected to LTDC_VSYNC

TIM_TIM4_ETR_TIM3_ETR
TIM4_ETR is connected to TIM3 ETR

TIM_TIM4_ETR_TIM5_ETR
TIM4_ETR is connected to TIM5 ETR

TIM_TIM4_ETR_DSI_TE
TIM2_ETR is connected to DSI_TE

TIM_TIM4_ETR_ADC2_AWD1
TIM4_ETR is connected to ADC2 AWD1

TIM_TIM4_ETR_ADC2_AWD2
TIM4_ETR is connected to ADC2 AWD2

TIM_TIM4_ETR_ADC2_AWD3
TIM4_ETR is connected to ADC2 AWD3

TIM_TIM4_ETR_DCMI_HSYNC
TIM4_ETR is connected to DCMI HSYNC

TIM_TIM4_ETR_LTDC_HSYNC
TIM4_ETR is connected to LTDC HSYNC

TIM_TIM5_ETR_GPIO
TIM5_ETR is not connected to I/O

TIM_TIM5_ETR_COMP1
TIM5_ETR is connected to COMP1 output

TIM_TIM5_ETR_COMP2
TIM5_ETR is connected to COMP2 output

TIM_TIM5_ETR_MSIK
TIM5_ETR is connected to MSIK

TIM_TIM5_ETR_HSI
TIM5_ETR is connected to HSI

TIM_TIM5_ETR_MSIS
TIM5_ETR is connected to MSIS

TIM_TIM5_ETR_DCMI_VSYNC
TIM5_ETR is connected to DCMI VSYNC

TIM_TIM5_ETR_LTDC_VSYNC
TIM5_ETR is connected to LTDC_VSYNC

TIM_TIM5_ETR_TIM2_ETR
TIM5_ETR is connected to TIM2 ETR

TIM_TIM5_ETR_TIM3_ETR
TIM5_ETR is connected to TIM3 ETR

TIM_TIM5_ETR_DSI_TE
TIM5_ETR is connected to DSI_TE

TIM_TIM5_ETR_DCMI_HSYNC
TIM5_ETR is connected to DCMI HSYNC

TIM_TIM5_ETR_LTDC_HSYNC
TIM5_ETR is connected to LTDC HSYNC

TIM_TIM8_ETR_GPIO
TIM8_ETR is not connected to I/O

TIM_TIM8_ETR_COMP1
TIM8_ETR is connected to COMP1 output

TIM_TIM8_ETR_COMP2
TIM8_ETR is connected to COMP2 output

TIM_TIM8_ETR_MSIK
TIM8_ETR is connected to MSIK

TIM_TIM8_ETR_HSI
TIM8_ETR is connected to HSI

TIM_TIM8_ETR_MSIS
TIM8_ETR is connected to MSIS

TIM_TIM8_ETR_ADC2_AWD2
TIM8_ETR is connected to ADC2 AWD2

TIM_TIM8_ETR_ADC2_AWD3
TIM8_ETR is connected to ADC2 AWD3

TIM_TIM8_ETR_ADC1_AWD1
TIM8_ETR is connected to ADC1 AWD1

TIM_TIM8_ETR_ADC1_AWD2
TIM8_ETR is connected to ADC1 AWD2

TIM_TIM8_ETR_ADC1_AWD3
TIM8_ETR is connected to ADC1 AWD3

TIM_TIM8_ETR_ADC4_AWD1
TIM8_ETR is connected to ADC4 AWD1

TIM_TIM8_ETR_ADC4_AWD2
TIM8_ETR is connected to ADC4 AWD2

TIM_TIM8_ETR_ADC4_AWD3
TIM8_ETR is connected to ADC4 AWD3

TIM_TIM8_ETR_ADC2_AWD1
TIM8_ETR is connected to ADC2 AWD1

TIM Extended Bitfield SMS preload enabling

TIM_SMS_PRELOAD_SOURCE_UPDATE
Prelaod of SMS bitfield is disabled

TIM_SMS_PRELOAD_SOURCE_INDEX
Preload of SMS bitfield is enabled

TIM Extended Timer input selection

TIM_TIM1_TI1_GPIO
TIM1_TI1 is connected to GPIO

TIM_TIM1_TI1_COMP1
TIM1_TI1 is connected to COMP1 OUT

TIM_TIM1_TI1_COMP2
TIM1_TI1 is connected to COMP2 OUT

TIM_TIM2_TI1_GPIO
TIM2_TI1 is connected to GPIO

TIM_TIM2_TI1_COMP1
TIM2_TI1 is connected to COMP1 OUT

TIM_TIM2_TI1_COMP2
TIM2_TI1 is connected to COMP2 OUT

TIM_TIM2_TI2_GPIO
TIM2_TI2 is connected to GPIO

TIM_TIM2_TI2_COMP1
TIM2_TI2 is connected to COMP1 OUT

TIM_TIM2_TI2_COMP2
TIM2_TI2 is connected to COMP2 OUT

TIM_TIM2_TI4_GPIO
TIM2_TI4 is connected to GPIO

TIM_TIM2_TI4_COMP1
TIM2_TI4 is connected to COMP1 OUT

TIM_TIM2_TI4_COMP2
TIM2_TI4 is connected to COMP2 OUT

TIM_TIM3_TI1_GPIO
TIM3_TI1 is connected to GPIO

TIM_TIM3_TI1_COMP1
TIM3_TI1 is connected to COMP1 OUT

TIM_TIM3_TI1_COMP2
TIM3_TI1 is connected to COMP2 OUT

TIM_TIM3_TI2_GPIO

TIM3_TI2 is connected to GPIO

TIM_TIM3_TI2_COMP1

TIM3_TI2 is connected to COMP1 OUT

TIM_TIM3_TI2_COMP2

TIM3_TI2 is connected to COMP2 OUT

TIM_TIM4_TI1_GPIO

TIM4_TI1 is connected to GPIO

TIM_TIM4_TI1_COMP1

TIM4_TI1 is connected to COMP1 OUT

TIM_TIM4_TI1_COMP2

TIM4_TI1 is connected to COMP2 OUT

TIM_TIM4_TI2_GPIO

TIM4_TI2 is connected to GPIO

TIM_TIM4_TI2_COMP1

TIM4_TI2 is connected to COMP1 OUT

TIM_TIM4_TI2_COMP2

TIM4_TI2 is connected to COMP2 OUT

TIM_TIM5_TI1_GPIO

TIM5_TI1 is connected to GPIO

TIM_TIM5_TI1_LSI

TIM5_TI1 is connected to LSI

TIM_TIM5_TI1_LSE

TIM5_TI1 is connected to LSE

TIM_TIM5_TI1_RTC_WKUP

TIM5_TI1 is connected to RTC Wakeup

TIM_TIM5_TI1_COMP1

TIM5_COMP1 is connected to COMP1 OUT

TIM_TIM5_TI1_COMP2

TIM5_COMP2 is connected to COMP2 OUT

TIM_TIM5_TI2_GPIO

TIM5_TI2 is connected to GPIO

TIM_TIM5_TI2_COMP1

TIM5_TI2 is connected to COMP1 OUT

TIM_TIM5_TI2_COMP2

TIM5_TI2 is connected to COMP2 OUT

TIM_TIM8_TI1_GPIO

TIM8_TI1 is connected to GPIO

TIM_TIM8_TI1_COMP1

TIM8_TI1 is connected to COMP1 OUT

TIM_TIM8_TI1_COMP2

TIM8_TI1 is connected to COMP2 OUT

TIM_TIM15_TI1_GPIO

TIM15_TI1 is connected to GPIO

TIM_TIM15_TI1_LSE

TIM15_TI1 is connected to LSE

TIM_TIM15_TI1_COMP1

TIM15_TI1 is connected to COMP1 OUT

TIM_TIM15_TI1_COMP2

TIM15_TI1 is connected to COMP2 OUT

TIM_TIM15_TI2_GPIO

TIM15_TI2 is connected to GPIO

TIM_TIM15_TI2_COMP2

TIM15_TI2 is connected to COMP2 OUT

TIM_TIM16_TI1_GPIO

TIM16_TI1 is connected to GPIO

TIM_TIM16_TI1_MCO

TIM16_TI1 is connected to MCO

TIM_TIM16_TI1_HSE_DIV32

TIM16_TI1 is connected to HSE/32

TIM_TIM16_TI1_RTC_WKUP

TIM16_TI1 is connected to RTC Wakeup

TIM_TIM16_TI1_LSE

TIM16_TI1 is connected to LSE

TIM_TIM16_TI1_LSI

TIM16_TI1 is connected to LSI

TIM_TIM16_TI1_MSIS_1024

TIM16_TI1 is connected to MSIS/1024

TIM_TIM16_TI1_MSIS_4

TIM16_TI1 is connected to MSIS/4

TIM_TIM16_TI1_HSI_256

TIM16_TI1 is connected to HSI/256

TIM_TIM17_TI1_GPIO

TIM17_TI1 is connected to GPIO

TIM_TIM17_TI1_MCO

TIM17_TI1 is connected to MCO

TIM_TIM17_TI1_HSE_DIV32
TIM17_TI1 is connected to HSE/32

TIM_TIM17_TI1_RTC_WKUP
TIM17_TI1 is connected to RTC Wakeup

TIM_TIM17_TI1_LSE
TIM17_TI1 is connected to LSE

TIM_TIM17_TI1_LSI
TIM17_TI1 is connected to LSI

TIM_TIM17_TI1_MSIS_1024
TIM17_TI1 is connected to MSIS/1024

TIM_TIM17_TI1_MSIS_4
TIM17_TI1 is connected to MSIS/4

TIM_TIM17_TI1_HSI_256
TIM17_TI1 is connected to HSI/256

83 HAL TSC Generic Driver

83.1 TSC Firmware driver registers structures

83.1.1 TSC_InitTypeDef

TSC_InitTypeDef is defined in the stm32u5xx_hal_tsc.h

Data Fields

- ***uint32_t CTPulseHighLength***
- ***uint32_t CTPulseLowLength***
- ***FunctionalState SpreadSpectrum***
- ***uint32_t SpreadSpectrumDeviation***
- ***uint32_t SpreadSpectrumPrescaler***
- ***uint32_t PulseGeneratorPrescaler***
- ***uint32_t MaxCountValue***
- ***uint32_t IODefaultMode***
- ***uint32_t SynchroPinPolarity***
- ***uint32_t AcquisitionMode***
- ***FunctionalState MaxCountInterrupt***
- ***uint32_t ChannelIOs***
- ***uint32_t ShieldIOs***
- ***uint32_t SamplingIOs***

Field Documentation

- ***uint32_t TSC_InitTypeDef::CTPulseHighLength***
  Charge-transfer high pulse length This parameter can be a value of ***TSC_CTPulseHL_Config***
- ***uint32_t TSC_InitTypeDef::CTPulseLowLength***
  Charge-transfer low pulse length This parameter can be a value of ***TSC_CTPulseLL_Config***
- ***FunctionalState TSC_InitTypeDef::SpreadSpectrum***
  Spread spectrum activation This parameter can be set to ENABLE or DISABLE.
- ***uint32_t TSC_InitTypeDef::SpreadSpectrumDeviation***
  Spread spectrum deviation This parameter must be a number between Min_Data = 0 and Max_Data = 127
- ***uint32_t TSC_InitTypeDef::SpreadSpectrumPrescaler***
  Spread spectrum prescaler This parameter can be a value of ***TSC_SpreadSpec_Prescaler***
- ***uint32_t TSC_InitTypeDef::PulseGeneratorPrescaler***
  Pulse generator prescaler This parameter can be a value of ***TSC_PulseGenerator_Prescaler***
- ***uint32_t TSC_InitTypeDef::MaxCountValue***
  Max count value This parameter can be a value of ***TSC_MaxCount_Value***
- ***uint32_t TSC_InitTypeDef::IODefaultMode***
  IO default mode This parameter can be a value of ***TSC_IO_Default_Mode***
- ***uint32_t TSC_InitTypeDef::SynchroPinPolarity***
  Synchro pin polarity This parameter can be a value of ***TSC_Synchro_Pin_Polarity***
- ***uint32_t TSC_InitTypeDef::AcquisitionMode***
  Acquisition mode This parameter can be a value of ***TSC_Acquisition_Mode***
- ***FunctionalState TSC_InitTypeDef::MaxCountInterrupt***
  Max count interrupt activation This parameter can be set to ENABLE or DISABLE.
- ***uint32_t TSC_InitTypeDef::ChannelIOs***
  Channel IOs mask
- ***uint32_t TSC_InitTypeDef::ShieldIOs***
  Shield IOs mask
- ***uint32_t TSC_InitTypeDef::SamplingIOs***
  Sampling IOs mask

83.1.2 TSC_IOConfigTypeDef

TSC_IOConfigTypeDef is defined in the stm32u5xx_hal_tsc.h

Data Fields

- ***uint32_t ChannelIOs***
- ***uint32_t ShieldIOs***
- ***uint32_t SamplingIOs***

Field Documentation

- ***uint32_t TSC_IOConfigTypeDef::ChannelIOs***
  Channel IOs mask
- ***uint32_t TSC_IOConfigTypeDef::ShieldIOs***
  Shield IOs mask
- ***uint32_t TSC_IOConfigTypeDef::SamplingIOs***
  Sampling IOs mask

83.1.3 __TSC_HandleTypeDef

__TSC_HandleTypeDef is defined in the stm32u5xx_hal_tsc.h

Data Fields

- ***TSC_TypeDef * Instance***
- ***TSC_InitTypeDef Init***
- ***__IO HAL_TSC_StateTypeDef State***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t ErrorCode***
- ***void(* ConvCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***TSC_TypeDef* __TSC_HandleTypeDef::Instance***
  Register base address
- ***TSC_InitTypeDef __TSC_HandleTypeDef::Init***
  Initialization parameters
- ***__IO HAL_TSC_StateTypeDef __TSC_HandleTypeDef::State***
  Peripheral state
- ***HAL_LockTypeDef __TSC_HandleTypeDef::Lock***
  Lock feature
- ***__IO uint32_t __TSC_HandleTypeDef::ErrorCode***
  TSC Error code
- ***void(* __TSC_HandleTypeDef::ConvCpltCallback)(struct __TSC_HandleTypeDef *htsc)***
  TSC Conversion complete callback
- ***void(* __TSC_HandleTypeDef::ErrorCallback)(struct __TSC_HandleTypeDef *htsc)***
  TSC Error callback
- ***void(* __TSC_HandleTypeDef::MspInitCallback)(struct __TSC_HandleTypeDef *htsc)***
  TSC Msp Init callback
- ***void(* __TSC_HandleTypeDef::MspDeInitCallback)(struct __TSC_HandleTypeDef *htsc)***
  TSC Msp DeInit callback

83.2 TSC Firmware driver API description

The following section lists the various functions of the TSC library.

83.2.1 TSC specific features

1. Proven and robust surface charge transfer acquisition principle
2. Supports up to 3 capacitive sensing channels per group

3. Capacitive sensing channels can be acquired in parallel offering a very good response time
4. Spread spectrum feature to improve system robustness in noisy environments
5. Full hardware management of the charge transfer acquisition sequence
6. Programmable charge transfer frequency
7. Programmable sampling capacitor I/O pin
8. Programmable channel I/O pin
9. Programmable max count value to avoid long acquisition when a channel is faulty
10. Dedicated end of acquisition and max count error flags with interrupt capability
11. One sampling capacitor for up to 3 capacitive sensing channels to reduce the system components
12. Compatible with proximity, touchkey, linear and rotary touch sensor implementation

83.2.2 How to use this driver

1. Enable the TSC interface clock using __HAL_RCC_TSC_CLK_ENABLE() macro.
2. GPIO pins configuration
   – Enable the clock for the TSC GPIOs using __HAL_RCC_GPIOx_CLK_ENABLE() macro.
   – Configure the TSC pins used as sampling IOs in alternate function output Open-Drain mode, and TSC pins used as channel/shield IOs in alternate function output Push-Pull mode using HAL_GPIO_Init() function.
3. Interrupts configuration
   – Configure the NVIC (if the interrupt model is used) using HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ() and function.
4. TSC configuration
   – Configure all TSC parameters and used TSC IOs using HAL_TSC_Init() function.

TSC peripheral alternate functions are mapped on AF9.

Acquisition sequence

- Discharge all IOs using HAL_TSC_IODischarge() function.
- Wait a certain time allowing a good discharge of all capacitors. This delay depends of the sampling capacitor and electrodes design.
- Select the channel IOs to be acquired using HAL_TSC_IOConfig() function.
- Launch the acquisition using either HAL_TSC_Start() or HAL_TSC_Start_IT() function. If the synchronized mode is selected, the acquisition will start as soon as the signal is received on the synchro pin.
- Wait the end of acquisition using either HAL_TSC_PollForAcquisition() or HAL_TSC_GetState() function or using WFI instruction for example.
- Check the group acquisition status using HAL_TSC_GroupGetStatus() function.
- Read the acquisition value using HAL_TSC_GroupGetValue() function.

Callback registration

The compilation flag USE_HAL_TSC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_TSC_RegisterCallback() to register an interrupt callback.

Function HAL_TSC_RegisterCallback() allows to register following callbacks:

- ConvCpltCallback : callback for conversion complete process.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_TSC_UnRegisterCallback to reset a callback to the default weak function. HAL_TSC_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID.

This function allows to reset following callbacks:

- ConvCpltCallback : callback for conversion complete process.

- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_TSC_Init() and when the state is HAL_TSC_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_TSC_ConvCpltCallback(), HAL_TSC_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_TSC_Init()/ HAL_TSC_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_TSC_Init()/ HAL_TSC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_TSC_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_TSC_STATE_READY or HAL_TSC_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/ DeInit. Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_TSC_RegisterCallback() before calling HAL_TSC_DeInit() or HAL_TSC_Init() function.

When the compilation flag USE_HAL_TSC_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and all callbacks are set to the corresponding weak functions.

83.2.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the TSC.
- De-initialize the TSC.

This section contains the following APIs:

- *HAL_TSC_Init()*
- *HAL_TSC_DeInit()*
- *HAL_TSC_MspInit()*
- *HAL_TSC_MspDeInit()*
- *HAL_TSC_RegisterCallback()*
- *HAL_TSC_UnRegisterCallback()*

83.2.4 IO Operation functions

This section provides functions allowing to:

- Start acquisition in polling mode.
- Start acquisition in interrupt mode.
- Stop conversion in polling mode.
- Stop conversion in interrupt mode.
- Poll for acquisition completed.
- Get group acquisition status.
- Get group acquisition value.

This section contains the following APIs:

- *HAL_TSC_Start()*
- *HAL_TSC_Start_IT()*
- *HAL_TSC_Stop()*
- *HAL_TSC_Stop_IT()*
- *HAL_TSC_PollForAcquisition()*
- *HAL_TSC_GroupGetStatus()*
- *HAL_TSC_GroupGetValue()*

83.2.5 Peripheral Control functions

This section provides functions allowing to:

- Configure TSC IOs
- Discharge TSC IOs

This section contains the following APIs:
- *HAL_TSC_IOConfig()*
- *HAL_TSC_IODischarge()*

83.2.6 State and Errors functions

This subsection provides functions allowing to
- Get TSC state.

This section contains the following APIs:
- *HAL_TSC_GetState()*

83.2.7 Detailed description of functions

HAL_TSC_Init

Function name

HAL_StatusTypeDef HAL_TSC_Init (TSC_HandleTypeDef * htsc)

Function description

Initialize the TSC peripheral according to the specified parameters in the TSC_InitTypeDef structure and initialize the associated handle.

Parameters

- **htsc:** TSC handle

Return values

- **HAL:** status

HAL_TSC_DeInit

Function name

HAL_StatusTypeDef HAL_TSC_DeInit (TSC_HandleTypeDef * htsc)

Function description

Deinitialize the TSC peripheral registers to their default reset values.

Parameters

- **htsc:** TSC handle

Return values

- **HAL:** status

HAL_TSC_MspInit

Function name

void HAL_TSC_MspInit (TSC_HandleTypeDef * htsc)

Function description

Initialize the TSC MSP.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

HAL_TSC_MspDeInit

Function name

void HAL_TSC_MspDeInit (TSC_HandleTypeDef * htsc)

Function description

DeInitialize the TSC MSP.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

HAL_TSC_RegisterCallback

Function name

HAL_StatusTypeDef HAL_TSC_RegisterCallback (TSC_HandleTypeDef * htsc, HAL_TSC_CallbackIDTypeDef CallbackID, pTSC_CallbackTypeDef pCallback)

Function description

Register a User TSC Callback To be used instead of the weak predefined callback.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_TSC_CONV_COMPLETE_CB_ID Conversion completed callback ID
  – HAL_TSC_ERROR_CB_ID Error callback ID
  – HAL_TSC_MSPINIT_CB_ID MspInit callback ID
  – HAL_TSC_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_TSC_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_TSC_UnRegisterCallback (TSC_HandleTypeDef * htsc, HAL_TSC_CallbackIDTypeDef CallbackID)

Function description

Unregister an TSC Callback TSC callback is redirected to the weak predefined callback.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values: This parameter can be one of the following values:
  – HAL_TSC_CONV_COMPLETE_CB_ID Conversion completed callback ID
  – HAL_TSC_ERROR_CB_ID Error callback ID
  – HAL_TSC_MSPINIT_CB_ID MspInit callback ID
  – HAL_TSC_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_TSC_Start

Function name

HAL_StatusTypeDef HAL_TSC_Start (TSC_HandleTypeDef * htsc)

Function description

Start the acquisition.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** status

HAL_TSC_Start_IT

Function name

HAL_StatusTypeDef HAL_TSC_Start_IT (TSC_HandleTypeDef * htsc)

Function description

Start the acquisition in interrupt mode.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** status.

HAL_TSC_Stop

Function name

HAL_StatusTypeDef HAL_TSC_Stop (TSC_HandleTypeDef * htsc)

Function description

Stop the acquisition previously launched in polling mode.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** status

HAL_TSC_Stop_IT

Function name

HAL_StatusTypeDef HAL_TSC_Stop_IT (TSC_HandleTypeDef * htsc)

Function description

Stop the acquisition previously launched in interrupt mode.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** status

HAL_TSC_PollForAcquisition

Function name

HAL_StatusTypeDef HAL_TSC_PollForAcquisition (TSC_HandleTypeDef * htsc)

Function description

Start acquisition and wait until completion.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** state

Notes

- There is no need of a timeout parameter as the max count error is already managed by the TSC peripheral.

HAL_TSC_GroupGetStatus

Function name

TSC_GroupStatusTypeDef HAL_TSC_GroupGetStatus (const TSC_HandleTypeDef * htsc, uint32_t gx_index)

Function description

Get the acquisition status for a group.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **gx_index:** Index of the group

Return values

- **Group:** status

HAL_TSC_GroupGetValue

Function name

uint32_t HAL_TSC_GroupGetValue (const TSC_HandleTypeDef * htsc, uint32_t gx_index)

Function description

Get the acquisition measure for a group.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **gx_index:** Index of the group

Return values

- **Acquisition:** measure

HAL_TSC_IOConfig

Function name

HAL_StatusTypeDef HAL_TSC_IOConfig (TSC_HandleTypeDef * htsc, const TSC_IOConfigTypeDef * config)

Function description

Configure TSC IOs.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **config:** Pointer to the configuration structure.

Return values

- **HAL:** status

HAL_TSC_IODischarge

Function name

HAL_StatusTypeDef HAL_TSC_IODischarge (TSC_HandleTypeDef * htsc, FunctionalState choice)

Function description

Discharge TSC IOs.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.
- **choice:** This parameter can be set to ENABLE or DISABLE.

Return values

- **HAL:** status

HAL_TSC_GetState

Function name

HAL_TSC_StateTypeDef HAL_TSC_GetState (TSC_HandleTypeDef * htsc)

Function description

Return the TSC handle state.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **HAL:** state

HAL_TSC_IRQHandler

Function name

void HAL_TSC_IRQHandler (TSC_HandleTypeDef * htsc)

Function description

Handle TSC interrupt request.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

HAL_TSC_ConvCpltCallback

Function name

void HAL_TSC_ConvCpltCallback (TSC_HandleTypeDef * htsc)

Function description

Acquisition completed callback in non-blocking mode.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

HAL_TSC_ErrorCallback

Function name

void HAL_TSC_ErrorCallback (TSC_HandleTypeDef * htsc)

Function description

Error callback in non-blocking mode.

Parameters

- **htsc:** Pointer to a TSC_HandleTypeDef structure that contains the configuration information for the specified TSC.

Return values

- **None:**

83.3 TSC Firmware driver defines

The following section lists the various define and macros of the module.

83.3.1 TSC

TSC

Acquisition Mode

TSC_ACQ_MODE_NORMAL

Normal acquisition mode (acquisition starts as soon as START bit is set)

TSC_ACQ_MODE_SYNCHRO

Synchronized acquisition mode (acquisition starts if START bit is set and when the selected signal is detected on the SYNC input pin)

CTPulse High Length

TSC_CTPH_1CYCLE

Charge transfer pulse high during 1 cycle (PGCLK)

TSC_CTPH_2CYCLES
Charge transfer pulse high during 2 cycles (PGCLK)

TSC_CTPH_3CYCLES
Charge transfer pulse high during 3 cycles (PGCLK)

TSC_CTPH_4CYCLES
Charge transfer pulse high during 4 cycles (PGCLK)

TSC_CTPH_5CYCLES
Charge transfer pulse high during 5 cycles (PGCLK)

TSC_CTPH_6CYCLES
Charge transfer pulse high during 6 cycles (PGCLK)

TSC_CTPH_7CYCLES
Charge transfer pulse high during 7 cycles (PGCLK)

TSC_CTPH_8CYCLES
Charge transfer pulse high during 8 cycles (PGCLK)

TSC_CTPH_9CYCLES
Charge transfer pulse high during 9 cycles (PGCLK)

TSC_CTPH_10CYCLES
Charge transfer pulse high during 10 cycles (PGCLK)

TSC_CTPH_11CYCLES
Charge transfer pulse high during 11 cycles (PGCLK)

TSC_CTPH_12CYCLES
Charge transfer pulse high during 12 cycles (PGCLK)

TSC_CTPH_13CYCLES
Charge transfer pulse high during 13 cycles (PGCLK)

TSC_CTPH_14CYCLES
Charge transfer pulse high during 14 cycles (PGCLK)

TSC_CTPH_15CYCLES
Charge transfer pulse high during 15 cycles (PGCLK)

TSC_CTPH_16CYCLES
Charge transfer pulse high during 16 cycles (PGCLK)

CTPulse Low Length

TSC_CTPL_1CYCLE
Charge transfer pulse low during 1 cycle (PGCLK)

TSC_CTPL_2CYCLES
Charge transfer pulse low during 2 cycles (PGCLK)

TSC_CTPL_3CYCLES
Charge transfer pulse low during 3 cycles (PGCLK)

TSC_CTPL_4CYCLES
Charge transfer pulse low during 4 cycles (PGCLK)

TSC_CTPL_5CYCLES

Charge transfer pulse low during 5 cycles (PGCLK)

TSC_CTPL_6CYCLES

Charge transfer pulse low during 6 cycles (PGCLK)

TSC_CTPL_7CYCLES

Charge transfer pulse low during 7 cycles (PGCLK)

TSC_CTPL_8CYCLES

Charge transfer pulse low during 8 cycles (PGCLK)

TSC_CTPL_9CYCLES

Charge transfer pulse low during 9 cycles (PGCLK)

TSC_CTPL_10CYCLES

Charge transfer pulse low during 10 cycles (PGCLK)

TSC_CTPL_11CYCLES

Charge transfer pulse low during 11 cycles (PGCLK)

TSC_CTPL_12CYCLES

Charge transfer pulse low during 12 cycles (PGCLK)

TSC_CTPL_13CYCLES

Charge transfer pulse low during 13 cycles (PGCLK)

TSC_CTPL_14CYCLES

Charge transfer pulse low during 14 cycles (PGCLK)

TSC_CTPL_15CYCLES

Charge transfer pulse low during 15 cycles (PGCLK)

TSC_CTPL_16CYCLES

Charge transfer pulse low during 16 cycles (PGCLK)

TSC Error Code definition

HAL_TSC_ERROR_NONE

No error

HAL_TSC_ERROR_INVALID_CALLBACK

Invalid Callback error

TSC Exported Macros

__HAL_TSC_RESET_HANDLE_STATE

Description:

- Reset TSC handle state.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_ENABLE

Description:
- Enable the TSC peripheral.

Parameters:
- __HANDLE__: TSC handle

Return value:
- None

__HAL_TSC_DISABLE

Description:
- Disable the TSC peripheral.

Parameters:
- __HANDLE__: TSC handle

Return value:
- None

__HAL_TSC_START_ACQ

Description:
- Start acquisition.

Parameters:
- __HANDLE__: TSC handle

Return value:
- None

__HAL_TSC_STOP_ACQ

Description:
- Stop acquisition.

Parameters:
- __HANDLE__: TSC handle

Return value:
- None

__HAL_TSC_SET_IODEF_OUTPPLOW

Description:
- Set IO default mode to output push-pull low.

Parameters:
- __HANDLE__: TSC handle

Return value:
- None

__HAL_TSC_SET_IODEF_INFLOAT

Description:
- Set IO default mode to input floating.

Parameters:
- __HANDLE__: TSC handle

Return value:
- None

__HAL_TSC_SET_SYNC_POL_FALL

Description:

- Set synchronization polarity to falling edge.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_SET_SYNC_POL_RISE_HIGH

Description:

- Set synchronization polarity to rising edge and high level.

Parameters:

- __HANDLE__: TSC handle

Return value:

- None

__HAL_TSC_ENABLE_IT

Description:

- Enable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_DISABLE_IT

Description:

- Disable TSC interrupt.

Parameters:

- __HANDLE__: TSC handle
- __INTERRUPT__: TSC interrupt

Return value:

- None

__HAL_TSC_GET_IT_SOURCE

Description:

- Check whether the specified TSC interrupt source is enabled or not.

Parameters:

- __HANDLE__: TSC Handle
- __INTERRUPT__: TSC interrupt

Return value:

- SET: or RESET

__HAL_TSC_GET_FLAG

Description:
- Check whether the specified TSC flag is set or not.

Parameters:
- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:
- SET: or RESET

__HAL_TSC_CLEAR_FLAG

Description:
- Clear the TSC's pending flag.

Parameters:
- __HANDLE__: TSC handle
- __FLAG__: TSC flag

Return value:
- None

__HAL_TSC_ENABLE_HYSTERESIS

Description:
- Enable schmitt trigger hysteresis on a group of IOs.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_DISABLE_HYSTERESIS

Description:
- Disable schmitt trigger hysteresis on a group of IOs.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_OPEN_ANALOG_SWITCH

Description:
- Open analog switch on a group of IOs.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_CLOSE_ANALOG_SWITCH

Description:
- Close analog switch on a group of IOs.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_ENABLE_CHANNEL

Description:
- Enable a group of IOs in channel mode.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_DISABLE_CHANNEL

Description:
- Disable a group of channel IOs.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_ENABLE_SAMPLING

Description:
- Enable a group of IOs in sampling mode.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_DISABLE_SAMPLING

Description:
- Disable a group of sampling IOs.

Parameters:
- __HANDLE__: TSC handle
- __GX_IOY_MASK__: IOs mask

Return value:
- None

__HAL_TSC_ENABLE_GROUP

Description:
- Enable acquisition groups.

Parameters:
- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:
- None

__HAL_TSC_DISABLE_GROUP

Description:
- Disable acquisition groups.

Parameters:
- __HANDLE__: TSC handle
- __GX_MASK__: Groups mask

Return value:
- None

__HAL_TSC_GET_GROUP_STATUS

Description:
- Gets acquisition group status.

Parameters:
- __HANDLE__: TSC Handle
- __GX_INDEX__: Group index

Return value:
- SET: or RESET

Flags definition

TSC_FLAG_EOA

End of acquisition flag

TSC_FLAG_MCE

Max count error flag

Group definition

TSC_GROUP1

TSC_GROUP2

TSC_GROUP3

TSC_GROUP4

TSC_GROUP5

TSC_GROUP6

TSC_GROUP7

TSC_GROUP8

TSC_GROUPX_NOT_SUPPORTED
TSC GroupX not supported

TSC_GROUP1_IO1
TSC Group1 IO1

TSC_GROUP1_IO2
TSC Group1 IO2

TSC_GROUP1_IO3
TSC Group1 IO3

TSC_GROUP1_IO4
TSC Group1 IO4

TSC_GROUP2_IO1
TSC Group2 IO1

TSC_GROUP2_IO2
TSC Group2 IO2

TSC_GROUP2_IO3
TSC Group2 IO3

TSC_GROUP2_IO4
TSC Group2 IO4

TSC_GROUP3_IO1
TSC Group3 IO1

TSC_GROUP3_IO2
TSC Group3 IO2

TSC_GROUP3_IO3
TSC Group3 IO3

TSC_GROUP3_IO4
TSC Group3 IO4

TSC_GROUP4_IO1
TSC Group4 IO1

TSC_GROUP4_IO2
TSC Group4 IO2

TSC_GROUP4_IO3
TSC Group4 IO3

TSC_GROUP4_IO4
TSC Group4 IO4

TSC_GROUP5_IO1
TSC Group5 IO1

TSC_GROUP5_IO2
TSC Group5 IO2

TSC_GROUP5_IO3

TSC Group5 IO3

TSC_GROUP5_IO4

TSC Group5 IO4

TSC_GROUP6_IO1

TSC Group6 IO1

TSC_GROUP6_IO2

TSC Group6 IO2

TSC_GROUP6_IO3

TSC Group6 IO3

TSC_GROUP6_IO4

TSC Group6 IO4

TSC_GROUP7_IO1

TSC Group7 IO1

TSC_GROUP7_IO2

TSC Group7 IO2

TSC_GROUP7_IO3

TSC Group7 IO3

TSC_GROUP7_IO4

TSC Group7 IO4

TSC_GROUP8_IO1

TSC Group8 IO1

TSC_GROUP8_IO2

TSC Group8 IO2

TSC_GROUP8_IO3

TSC Group8 IO3

TSC_GROUP8_IO4

TSC Group8 IO4

Interrupts definition

TSC_IT_EOA

End of acquisition interrupt enable

TSC_IT_MCE

Max count error interrupt enable

IO Default Mode

TSC_IODEF_OUT_PP_LOW

I/Os are forced to output push-pull low

TSC_IODEF_IN_FLOAT

I/Os are in input floating

Max Count Value

TSC_MCV_255

255 maximum number of charge transfer pulses

TSC_MCV_511

511 maximum number of charge transfer pulses

TSC_MCV_1023

1023 maximum number of charge transfer pulses

TSC_MCV_2047

2047 maximum number of charge transfer pulses

TSC_MCV_4095

4095 maximum number of charge transfer pulses

TSC_MCV_8191

8191 maximum number of charge transfer pulses

TSC_MCV_16383

16383 maximum number of charge transfer pulses

Pulse Generator Prescaler

TSC_PG_PRESC_DIV1

Pulse Generator HCLK Div1

TSC_PG_PRESC_DIV2

Pulse Generator HCLK Div2

TSC_PG_PRESC_DIV4

Pulse Generator HCLK Div4

TSC_PG_PRESC_DIV8

Pulse Generator HCLK Div8

TSC_PG_PRESC_DIV16

Pulse Generator HCLK Div16

TSC_PG_PRESC_DIV32

Pulse Generator HCLK Div32

TSC_PG_PRESC_DIV64

Pulse Generator HCLK Div64

TSC_PG_PRESC_DIV128

Pulse Generator HCLK Div128

Spread Spectrum Prescaler

TSC_SS_PRESC_DIV1

Spread Spectrum Prescaler Div1

TSC_SS_PRESC_DIV2

Spread Spectrum Prescaler Div2

Synchro Pin Polarity

TSC_SYNC_POLARITY_FALLING

Falling edge only

TSC_SYNC_POLARITY_RISING

Rising edge and high level

84 HAL UART Generic Driver

84.1 UART Firmware driver registers structures

84.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32u5xx_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
  This member configures the UART communication baud rate. The baud rate register is computed using the following formula: LPUART: ======= Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where lpuart_ker_ck_pres is the UART input clock divided by a prescaler UART: =====
  – If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))
  – If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where uart_ker_ck_pres is the UART input clock divided by a prescaler
- ***uint32_t UART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.
- ***uint32_t UART_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.
- ***uint32_t UART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

84.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32u5xx_hal_uart.h

Data Fields

- *uint32_t AdvFeatureInit*
- *uint32_t TxPinLevelInvert*
- *uint32_t RxPinLevelInvert*
- *uint32_t DataInvert*
- *uint32_t Swap*
- *uint32_t OverrunDisable*
- *uint32_t DMADisableonRxError*
- *uint32_t AutoBaudRateEnable*
- *uint32_t AutoBaudRateMode*
- *uint32_t MSBFirst*

Field Documentation

- *uint32_t UART_AdvFeatureInitTypeDef::AdvFeatureInit*
  Specifies which advanced UART features is initialized. Several Advanced Features may be initialized at the same time . This parameter can be a value of *UART_Advanced_Features_Initialization_Type*.
- *uint32_t UART_AdvFeatureInitTypeDef::TxPinLevelInvert*
  Specifies whether the TX pin active level is inverted. This parameter can be a value of *UART_Tx_Inv*.
- *uint32_t UART_AdvFeatureInitTypeDef::RxPinLevelInvert*
  Specifies whether the RX pin active level is inverted. This parameter can be a value of *UART_Rx_Inv*.
- *uint32_t UART_AdvFeatureInitTypeDef::DataInvert*
  Specifies whether data are inverted (positive/direct logic vs negative/inverted logic). This parameter can be a value of *UART_Data_Inv*.
- *uint32_t UART_AdvFeatureInitTypeDef::Swap*
  Specifies whether TX and RX pins are swapped. This parameter can be a value of *UART_Rx_Tx_Swap*.
- *uint32_t UART_AdvFeatureInitTypeDef::OverrunDisable*
  Specifies whether the reception overrun detection is disabled. This parameter can be a value of *UART_Overrun_Disable*.
- *uint32_t UART_AdvFeatureInitTypeDef::DMADisableonRxError*
  Specifies whether the DMA is disabled in case of reception error. This parameter can be a value of *UART_DMA_Disable_on_Rx_Error*.
- *uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateEnable*
  Specifies whether auto Baud rate detection is enabled. This parameter can be a value of *UART_AutoBaudRate_Enable*.
- *uint32_t UART_AdvFeatureInitTypeDef::AutoBaudRateMode*
  If auto Baud rate detection is enabled, specifies how the rate detection is carried out. This parameter can be a value of *UART_AutoBaud_Rate_Mode*.
- *uint32_t UART_AdvFeatureInitTypeDef::MSBFirst*
  Specifies whether MSB is sent first on UART line. This parameter can be a value of *UART_MSB_First*.

84.1.3 __UART_HandleTypeDef

__UART_HandleTypeDef is defined in the stm32u5xx_hal_uart.h

Data Fields

- *USART_TypeDef * Instance*
- *UART_InitTypeDef Init*
- *UART_AdvFeatureInitTypeDef AdvancedInit*
- *const uint8_t * pTxBuffPtr*
- *uint16_t TxXferSize*
- *__IO uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*

- *uint16_t Mask*
- *uint32_t FifoMode*
- *uint16_t NbRxDataToProcess*
- *uint16_t NbTxDataToProcess*
- *__IO HAL_UART_RxTypeTypeDef ReceptionType*
- *__IO HAL_UART_RxEventTypeTypeDef RxEventType*
- *void(* RxISR*
- *void(* TxISR*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_UART_StateTypeDef gState*
- *__IO HAL_UART_StateTypeDef RxState*
- *__IO uint32_t ErrorCode*
- *void(* TxHalfCpltCallback*
- *void(* TxCpltCallback*
- *void(* RxHalfCpltCallback*
- *void(* RxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* AbortTransmitCpltCallback*
- *void(* AbortReceiveCpltCallback*
- *void(* RxFifoFullCallback*
- *void(* TxFifoEmptyCallback*
- *void(* RxEventCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***USART_TypeDef* __UART_HandleTypeDef::Instance***
  UART registers base address
- ***UART_InitTypeDef __UART_HandleTypeDef::Init***
  UART communication parameters
- ***UART_AdvFeatureInitTypeDef __UART_HandleTypeDef::AdvancedInit***
  UART Advanced Features initialization parameters
- ***const uint8_t* __UART_HandleTypeDef::pTxBuffPtr***
  Pointer to UART Tx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::TxXferSize***
  UART Tx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::TxXferCount***
  UART Tx Transfer Counter
- ***uint8_t* __UART_HandleTypeDef::pRxBuffPtr***
  Pointer to UART Rx transfer Buffer
- ***uint16_t __UART_HandleTypeDef::RxXferSize***
  UART Rx Transfer size
- ***__IO uint16_t __UART_HandleTypeDef::RxXferCount***
  UART Rx Transfer Counter
- ***uint16_t __UART_HandleTypeDef::Mask***
  UART Rx RDR register mask
- ***uint32_t __UART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode is being used. This parameter can be a value of ***UARTEx_FIFO_mode***.
- ***uint16_t __UART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution

- ***uint16_t __UART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***__IO HAL_UART_RxTypeTypeDef __UART_HandleTypeDef::ReceptionType***
  Type of ongoing reception
- ***__IO HAL_UART_RxEventTypeTypeDef __UART_HandleTypeDef::RxEventType***
  Type of Rx Event
- ***void(* __UART_HandleTypeDef::RxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Rx IRQ handler
- ***void(* __UART_HandleTypeDef::TxISR)(struct __UART_HandleTypeDef *huart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmatx***
  UART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __UART_HandleTypeDef::hdmarx***
  UART Rx DMA Handle parameters
- ***HAL_LockTypeDef __UART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::gState***
  UART state information related to global Handle management and also related to Tx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO HAL_UART_StateTypeDef __UART_HandleTypeDef::RxState***
  UART state information related to Rx operations. This parameter can be a value of **HAL_UART_StateTypeDef**
- ***__IO uint32_t __UART_HandleTypeDef::ErrorCode***
  UART Error code
- ***void(* __UART_HandleTypeDef::TxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Half Complete Callback
- ***void(* __UART_HandleTypeDef::TxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Complete Callback
- ***void(* __UART_HandleTypeDef::RxHalfCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Half Complete Callback
- ***void(* __UART_HandleTypeDef::RxCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Complete Callback
- ***void(* __UART_HandleTypeDef::ErrorCallback)(struct __UART_HandleTypeDef *huart)***
  UART Error Callback
- ***void(* __UART_HandleTypeDef::AbortCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Complete Callback
- ***void(* __UART_HandleTypeDef::AbortTransmitCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Transmit Complete Callback
- ***void(* __UART_HandleTypeDef::AbortReceiveCpltCallback)(struct __UART_HandleTypeDef *huart)***
  UART Abort Receive Complete Callback
- ***void(* __UART_HandleTypeDef::RxFifoFullCallback)(struct __UART_HandleTypeDef *huart)***
  UART Rx Fifo Full Callback
- ***void(* __UART_HandleTypeDef::TxFifoEmptyCallback)(struct __UART_HandleTypeDef *huart)***
  UART Tx Fifo Empty Callback
- ***void(* __UART_HandleTypeDef::RxEventCallback)(struct __UART_HandleTypeDef *huart, uint16_t Pos)***
  UART Reception Event Callback
- ***void(* __UART_HandleTypeDef::MspInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp Init callback
- ***void(* __UART_HandleTypeDef::MspDeInitCallback)(struct __UART_HandleTypeDef *huart)***
  UART Msp DeInit callback

84.2 UART Firmware driver API description

The following section lists the various functions of the UART library.

84.2.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
   - Enable the USARTx interface clock.
   - UART pins configuration:
     - Enable the clock for the UART GPIOs.
     - Configure these UART pins as alternate function pull-up.
   - NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
     - Configure the USARTx interrupt priority.
     - Enable the NVIC USART IRQ handle.
   - UART interrupts handling:

Note: The specific UART interrupts (Transmission complete interrupt, RXNE interrupt, RX/TX FIFOs related interrupts and Error Interrupts) are managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.

   - DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
     - Declare a DMA handle structure for the Tx/Rx channel.
     - Enable the DMAx interface clock.
     - Configure the declared DMA handle structure with the required Tx/Rx parameters.
     - Configure the DMA Tx/Rx channel.
     - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
     - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Prescaler value , Hardware flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.
4. If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...) in the huart handle AdvancedInit structure.
5. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
6. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
7. For the UART LIN (Local Interconnection Network) mode, initialize the UART registers by calling the HAL_LIN_Init() API.
8. For the UART Multiprocessor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.
9. For the UART RS485 Driver Enabled mode, initialize the UART registers by calling the HAL_RS485Ex_Init() API.

Note: These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(), also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

84.2.2 Callback registration

The compilation define USE_HAL_UART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_UART_RegisterCallback() to register a user callback. Function HAL_UART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.

- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_UART_UnRegisterCallback() to reset a callback to the default weak function. HAL_UART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- AbortTransmitCpltCallback : Abort Transmit Complete Callback.
- AbortReceiveCpltCallback : Abort Receive Complete Callback.
- WakeupCallback : Wakeup Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : UART MspInit.
- MspDeInitCallback : UART MspDeInit.

For specific callback RxEventCallback, use dedicated registration/reset functions: respectively HAL_UART_RegisterRxEventCallback() , HAL_UART_UnRegisterRxEventCallback().

By default, after the HAL_UART_Init() and when the state is HAL_UART_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_UART_TxCpltCallback(), HAL_UART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_UART_Init() and HAL_UART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_UART_Init() and HAL_UART_DeInit() keep and use the user MspInit/ MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_UART_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_UART_STATE_READY or HAL_UART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_UART_RegisterCallback() before calling HAL_UART_DeInit() or HAL_UART_Init() function.

When The compilation define USE_HAL_UART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

84.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method

- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init()and HAL_MultiProcessor_Init()API follow respectively the UART asynchronous, UART Half duplex, UART LIN mode and UART multiprocessor mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:
- *HAL_UART_Init()*
- *HAL_HalfDuplex_Init()*
- *HAL_LIN_Init()*
- *HAL_MultiProcessor_Init()*
- *HAL_UART_DeInit()*
- *HAL_UART_MspInit()*
- *HAL_UART_MspDeInit()*
- *HAL_UART_RegisterCallback()*
- *HAL_UART_UnRegisterCallback()*
- *HAL_UART_RegisterRxEventCallback()*
- *HAL_UART_UnRegisterRxEventCallback()*

84.2.4 IO operation functions

This section contains the following APIs:
- *HAL_UART_Transmit()*
- *HAL_UART_Receive()*
- *HAL_UART_Transmit_IT()*
- *HAL_UART_Receive_IT()*
- *HAL_UART_Transmit_DMA()*
- *HAL_UART_Receive_DMA()*
- *HAL_UART_DMAPause()*
- *HAL_UART_DMAResume()*
- *HAL_UART_DMAStop()*
- *HAL_UART_Abort()*
- *HAL_UART_AbortTransmit()*
- *HAL_UART_AbortReceive()*
- *HAL_UART_Abort_IT()*
- *HAL_UART_AbortTransmit_IT()*
- *HAL_UART_AbortReceive_IT()*
- *HAL_UART_IRQHandler()*
- *HAL_UART_TxCpltCallback()*
- *HAL_UART_TxHalfCpltCallback()*
- *HAL_UART_RxCpltCallback()*
- *HAL_UART_RxHalfCpltCallback()*
- *HAL_UART_ErrorCallback()*
- *HAL_UART_AbortCpltCallback()*
- *HAL_UART_AbortTransmitCpltCallback()*
- *HAL_UART_AbortReceiveCpltCallback()*

- *HAL_UARTEx_RxEventCallback()*

84.2.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART.
- HAL_UART_ReceiverTimeout_Config() API allows to configure the receiver timeout value on the fly
- HAL_UART_EnableReceiverTimeout() API enables the receiver timeout feature
- HAL_UART_DisableReceiverTimeout() API disables the receiver timeout feature
- HAL_MultiProcessor_EnableMuteMode() API enables mute mode
- HAL_MultiProcessor_DisableMuteMode() API disables mute mode
- HAL_MultiProcessor_EnterMuteMode() API enters mute mode
- UART_SetConfig() API configures the UART peripheral
- UART_AdvFeatureConfig() API optionally configures the UART advanced features
- UART_CheckIdleState() API ensures that TEACK and/or REACK are set after initialization
- HAL_HalfDuplex_EnableTransmitter() API disables receiver and enables transmitter
- HAL_HalfDuplex_EnableReceiver() API disables transmitter and enables receiver
- HAL_LIN_SendBreak() API transmits the break characters

This section contains the following APIs:
- *HAL_UART_ReceiverTimeout_Config()*
- *HAL_UART_EnableReceiverTimeout()*
- *HAL_UART_DisableReceiverTimeout()*
- *HAL_MultiProcessor_EnableMuteMode()*
- *HAL_MultiProcessor_DisableMuteMode()*
- *HAL_MultiProcessor_EnterMuteMode()*
- *HAL_HalfDuplex_EnableTransmitter()*
- *HAL_HalfDuplex_EnableReceiver()*
- *HAL_LIN_SendBreak()*

84.2.6 Peripheral State and Error functions

This subsection provides functions allowing to :
- Return the UART handle state.
- Return the UART handle error code

This section contains the following APIs:
- *HAL_UART_GetState()*
- *HAL_UART_GetError()*

84.2.7 Detailed description of functions

HAL_UART_Init

Function name

HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)

Function description

Initialize the UART mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_Init

Function name

HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)

Function description

Initialize the half-duplex mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_LIN_Init

Function name

HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)

Function description

Initialize the LIN mode according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **BreakDetectLength:** Specifies the LIN break detection length. This parameter can be one of the following values:
  – UART_LINBREAKDETECTLENGTH_10B 10-bit break detection
  – UART_LINBREAKDETECTLENGTH_11B 11-bit break detection

Return values

- **HAL:** status

HAL_MultiProcessor_Init

Function name

HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)

Function description

Initialize the multiprocessor mode according to the specified parameters in the UART_InitTypeDef and initialize the associated handle.

Parameters

- **huart:** UART handle.
- **Address:** UART node address (4-, 6-, 7- or 8-bit long).
- **WakeUpMethod:** Specifies the UART wakeup method. This parameter can be one of the following values:
  – UART_WAKEUPMETHOD_IDLELINE WakeUp by an idle line detection
  – UART_WAKEUPMETHOD_ADDRESSMARK WakeUp by an address mark

Return values

- **HAL:** status

Notes

- If the user resorts to idle line detection wake up, the Address parameter is useless and ignored by the initialization function.
- If the user resorts to address mark wake up, the address length detection is configured by default to 4 bits only. For the UART to be able to manage 6-, 7- or 8-bit long addresses detection, the API HAL_MultiProcessorEx_AddressLength_Set() must be called after HAL_MultiProcessor_Init().

HAL_UART_DeInit

Function name

HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_MspInit

Function name

void HAL_UART_MspInit (UART_HandleTypeDef * huart)

Function description

Initialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_MspDeInit

Function name

void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)

Function description

DeInitialize the UART MSP.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID, pUART_CallbackTypeDef pCallback)

Function description

Register a User UART Callback To be used to override the weak predefined callback.

Parameters

- **huart:** uart handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_UART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_UART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_UART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_UART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_UART_ERROR_CB_ID Error Callback ID
  - HAL_UART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_UART_WAKEUP_CB_ID Wakeup Callback ID
  - HAL_UART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_UART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_UART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_UART_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

Notes

- The HAL_UART_RegisterCallback() may be called before HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init() or HAL_RS485Ex_Init() in HAL_UART_STATE_RESET to register callbacks for HAL_UART_MSPINIT_CB_ID and HAL_UART_MSPDEINIT_CB_ID

HAL_UART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterCallback (UART_HandleTypeDef * huart, HAL_UART_CallbackIDTypeDef CallbackID)

Function description

Unregister an UART Callback UART callaback is redirected to the weak predefined callback.

Parameters

- **huart:** uart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_UART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_UART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_UART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_UART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_UART_ERROR_CB_ID Error Callback ID
  - HAL_UART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_UART_ABORT_TRANSMIT_COMPLETE_CB_ID Abort Transmit Complete Callback ID
  - HAL_UART_ABORT_RECEIVE_COMPLETE_CB_ID Abort Receive Complete Callback ID
  - HAL_UART_WAKEUP_CB_ID Wakeup Callback ID
  - HAL_UART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_UART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_UART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_UART_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_UART_UnRegisterCallback() may be called before HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init() or HAL_RS485Ex_Init() in HAL_UART_STATE_RESET to un-register callbacks for HAL_UART_MSPINIT_CB_ID and HAL_UART_MSPDEINIT_CB_ID

HAL_UART_RegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_RegisterRxEventCallback (UART_HandleTypeDef * huart, pUART_RxEventCallbackTypeDef pCallback)

Function description

Register a User UART Rx Event Callback To be used instead of the weak predefined callback.

Parameters

- **huart:** Uart handle
- **pCallback:** Pointer to the Rx Event Callback function

Return values

- **HAL:** status

HAL_UART_UnRegisterRxEventCallback

Function name

HAL_StatusTypeDef HAL_UART_UnRegisterRxEventCallback (UART_HandleTypeDef * huart)

Function description

UnRegister the UART Rx Event Callback UART Rx Event Callback is redirected to the weak HAL_UARTEx_RxEventCallback() predefined callback.

Parameters

- **huart:** Uart handle

Return values

- **HAL:** status

HAL_UART_Transmit

Function name

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Send an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.
- When FIFO mode is enabled, writing a data in the TDR register adds one data to the TXFIFO. Write operations to the TDR register are performed when TXFNF flag is set. From hardware perspective, TXFNF flag and TXE are mapped on the same bit-field.

HAL_UART_Receive

Function name

HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_IT

Function name

HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, const uint8_t * pData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pData.

HAL_UART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pData.

HAL_UART_DMAPause

Function name

HAL_StatusTypeDef HAL_UART_DMAPause (UART_HandleTypeDef * huart)

Function description

Pause the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAResume

Function name

HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)

Function description

Resume the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_DMAStop

Function name

HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)

Function description

Stop the DMA Transfer.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UART_Abort

Function name

HAL_StatusTypeDef HAL_UART_Abort (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortTransmit

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_AbortReceive

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (blocking mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_UART_Abort_IT

Function name

HAL_StatusTypeDef HAL_UART_Abort_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortTransmit_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortTransmit_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Transmit transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Tx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Tx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_AbortReceive_IT

Function name

HAL_StatusTypeDef HAL_UART_AbortReceive_IT (UART_HandleTypeDef * huart)

Function description

Abort ongoing Receive transfer (Interrupt mode).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing Rx transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable UART Interrupts (Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_UART_IRQHandler

Function name

void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)

Function description

Handle UART interrupt request.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxHalfCpltCallback

Function name

void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_TxCpltCallback

Function name

void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)

Function description

Tx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxHalfCpltCallback

Function name

void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Half Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_RxCpltCallback

Function name

void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

Function description

Rx Transfer completed callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_ErrorCallback

Function name

void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)

Function description

UART error callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortCpltCallback

Function name

void HAL_UART_AbortCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortTransmitCpltCallback

Function name

void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UART_AbortReceiveCpltCallback

Function name

void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef * huart)

Function description

UART Abort Receive Complete callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_RxEventCallback

Function name

void HAL_UARTEx_RxEventCallback (UART_HandleTypeDef * huart, uint16_t Size)

Function description

Reception Event Callback (Rx event notification called after use of advanced reception service).

Parameters

- **huart:** UART handle
- **Size:** Number of data available in application reception buffer (indicates a position in reception buffer until which, data are available)

Return values

- **None:**

HAL_UART_ReceiverTimeout_Config

Function name

void HAL_UART_ReceiverTimeout_Config (UART_HandleTypeDef * huart, uint32_t TimeoutValue)

Function description

Update on the fly the receiver timeout value in RTOR register.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
- **TimeoutValue:** receiver timeout value in number of baud blocks. The timeout value must be less or equal to 0x0FFFFFFFF.

Return values

- **None:**

HAL_UART_EnableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_EnableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_UART_DisableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_DisableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Disable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

Transmit break characters.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** UART handle.

Return values

- **None:**

Notes

- To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

Enable the UART transmitter and disable the UART receiver.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver and disable the UART transmitter.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status.

HAL_UART_GetState

Function name

HAL_UART_StateTypeDef HAL_UART_GetState (const UART_HandleTypeDef * huart)

Function description

Return the UART handle state.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **HAL:** state

HAL_UART_GetError

Function name

uint32_t HAL_UART_GetError (const UART_HandleTypeDef * huart)

Function description

Return the UART handle error code.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **UART:** Error Code

UART_InitCallbacksToDefault

Function name

void UART_InitCallbacksToDefault (UART_HandleTypeDef * huart)

Function description

Initialize the callbacks to their default values.

Parameters

- **huart:** UART handle.

Return values

- **none:**

UART_SetConfig

Function name

HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_CheckIdleState

Function name

HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)

Function description

Check the UART Idle State.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name

HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)

Function description

This function handles UART Communication Timeout.

Parameters

- **huart:** UART handle.
- **Flag:** Specifies the UART flag to check
- **Status:** The actual Flag status (SET or RESET)
- **Tickstart:** Tick start value
- **Timeout:** Timeout duration

Return values

- **HAL:** status

UART_AdvFeatureConfig

Function name

void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral advanced features.

Parameters

- **huart:** UART handle.

Return values

- **None:**

UART_Start_Receive_IT

Function name

HAL_StatusTypeDef UART_Start_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in Interrupt mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

UART_Start_Receive_DMA

Function name

HAL_StatusTypeDef UART_Start_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in DMA mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

84.3 UART Firmware driver defines

The following section lists the various define and macros of the module.

84.3.1 UART

UART

UART Advanced Feature Initialization Type

UART_ADVFEATURE_NO_INIT

No advanced feature initialization

UART_ADVFEATURE_TXINVERT_INIT

TX pin active level inversion

UART_ADVFEATURE_RXINVERT_INIT

RX pin active level inversion

UART_ADVFEATURE_DATAINVERT_INIT

Binary data inversion

UART_ADVFEATURE_SWAP_INIT

TX/RX pins swap

UART_ADVFEATURE_RXOVERRUNDISABLE_INIT

RX overrun disable

UART_ADVFEATURE_DMADISABLEONERROR_INIT

DMA disable on Reception Error

UART_ADVFEATURE_AUTOBAUDRATE_INIT

Auto Baud rate detection initialization

UART_ADVFEATURE_MSBFIRST_INIT

Most significant bit sent/received first

UART Advanced Feature Auto BaudRate Enable

UART_ADVFEATURE_AUTOBAUDRATE_DISABLE

RX Auto Baud rate detection enable

UART_ADVFEATURE_AUTOBAUDRATE_ENABLE

RX Auto Baud rate detection disable

UART Advanced Feature AutoBaud Rate Mode

UART_ADVFEATURE_AUTOBAUDRATE_ONSTARTBIT

Auto Baud rate detection on start bit

UART_ADVFEATURE_AUTOBAUDRATE_ONFALLINGEDGE

Auto Baud rate detection on falling edge

UART_ADVFEATURE_AUTOBAUDRATE_ON0X7FFRAME

Auto Baud rate detection on 0x7F frame detection

UART_ADVFEATURE_AUTOBAUDRATE_ON0X55FRAME

Auto Baud rate detection on 0x55 frame detection

UART Clock Prescaler

UART_PRESCALER_DIV1

fclk_pres = fclk

UART_PRESCALER_DIV2

fclk_pres = fclk/2

UART_PRESCALER_DIV4

fclk_pres = fclk/4

UART_PRESCALER_DIV6

fclk_pres = fclk/6

UART_PRESCALER_DIV8

fclk_pres = fclk/8

UART_PRESCALER_DIV10

fclk_pres = fclk/10

UART_PRESCALER_DIV12

fclk_pres = fclk/12

UART_PRESCALER_DIV16

fclk_pres = fclk/16

UART_PRESCALER_DIV32

fclk_pres = fclk/32

UART_PRESCALER_DIV64

fclk_pres = fclk/64

UART_PRESCALER_DIV128

fclk_pres = fclk/128

UART_PRESCALER_DIV256

fclk_pres = fclk/256

UART Driver Enable Assertion Time LSB Position In CR1 Register

UART_CR1_DEAT_ADDRESS_LSB_POS

UART Driver Enable assertion time LSB position in CR1 register

UART Driver Enable DeAssertion Time LSB Position In CR1 Register

UART_CR1_DEDT_ADDRESS_LSB_POS

UART Driver Enable de-assertion time LSB position in CR1 register

UART Address-matching LSB Position In CR2 Register

UART_CR2_ADDRESS_LSB_POS

UART address-matching LSB position in CR2 register

UART Advanced Feature Binary Data Inversion

UART_ADVFEATURE_DATAINV_DISABLE

Binary data inversion disable

UART_ADVFEATURE_DATAINV_ENABLE

Binary data inversion enable

UART Advanced Feature DMA Disable On Rx Error

UART_ADVFEATURE_DMA_ENABLEONRXERROR

DMA enable on Reception Error

UART_ADVFEATURE_DMA_DISABLEONRXERROR

DMA disable on Reception Error

UART DMA Rx

UART_DMA_RX_DISABLE

UART DMA RX disabled

UART_DMA_RX_ENABLE

UART DMA RX enabled

UART DMA Tx

UART_DMA_TX_DISABLE

UART DMA TX disabled

UART_DMA_TX_ENABLE

UART DMA TX enabled

UART DriverEnable Polarity

UART_DE_POLARITY_HIGH

Driver enable signal is active high

UART_DE_POLARITY_LOW

Driver enable signal is active low

UART Error Definition

HAL_UART_ERROR_NONE

No error

HAL_UART_ERROR_PE

Parity error

HAL_UART_ERROR_NE

Noise error

HAL_UART_ERROR_FE

Frame error

HAL_UART_ERROR_ORE

Overrun error

HAL_UART_ERROR_DMA

DMA transfer error

HAL_UART_ERROR_RTO

Receiver Timeout error

HAL_UART_ERROR_INVALID_CALLBACK

Invalid Callback error

UART Exported Macros

__HAL_UART_RESET_HANDLE_STATE

Description:

- Reset UART handle states.

Parameters:

- __HANDLE__: UART handle.

Return value:

- None

__HAL_UART_FLUSH_DRREGISTER

Description:

- Flush the UART Data registers.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FLAG

Description:

- Clear the specified UART pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_TXFECF TXFIFO empty clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_RTOF Receiver Timeout clear flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag

Return value:

- None

__HAL_UART_CLEAR_PEFLAG

Description:

- Clear the UART PE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_FEFLAG

Description:

- Clear the UART FE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_NEFLAG

Description:

- Clear the UART NE pending flag.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

__HAL_UART_CLEAR_OREFLAG

Description:
- Clear the UART ORE pending flag.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_CLEAR_IDLEFLAG

Description:
- Clear the UART IDLE pending flag.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_CLEAR_TXFECF

Description:
- Clear the UART TX FIFO empty clear flag.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_GET_FLAG

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - UART_FLAG_TXFT TXFIFO threshold flag
  - UART_FLAG_RXFT RXFIFO threshold flag
  - UART_FLAG_RXFF RXFIFO Full flag
  - UART_FLAG_TXFE TXFIFO Empty flag
  - UART_FLAG_REACK Receive enable acknowledge flag
  - UART_FLAG_TEACK Transmit enable acknowledge flag
  - UART_FLAG_RWU Receiver wake up flag (if the UART in mute mode)
  - UART_FLAG_SBKF Send Break flag
  - UART_FLAG_CMF Character match flag
  - UART_FLAG_BUSY Busy flag
  - UART_FLAG_ABRF Auto Baud rate detection flag
  - UART_FLAG_ABRE Auto Baud rate detection error flag
  - UART_FLAG_CTS CTS Change flag
  - UART_FLAG_LBDF LIN Break detection flag
  - UART_FLAG_TXE Transmit data register empty flag
  - UART_FLAG_TXFNF UART TXFIFO not full flag
  - UART_FLAG_TC Transmission Complete flag
  - UART_FLAG_RXNE Receive data register not empty flag
  - UART_FLAG_RXFNE UART RXFIFO not empty flag
  - UART_FLAG_RTOF Receiver Timeout flag
  - UART_FLAG_IDLE Idle Line detection flag
  - UART_FLAG_ORE Overrun Error flag
  - UART_FLAG_NE Noise Error flag
  - UART_FLAG_FE Framing Error flag
  - UART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_UART_ENABLE_IT

Description:

- Enable the specified UART interrupt.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to enable. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (frame error, noise error, overrun error)

Return value:

- None

__HAL_UART_DISABLE_IT

Description:
- Disable the specified UART interrupt.

Parameters:
- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to disable. This parameter can be one of the following values:
  – UART_IT_RXFF RXFIFO Full interrupt
  – UART_IT_TXFE TXFIFO Empty interrupt
  – UART_IT_RXFT RXFIFO threshold interrupt
  – UART_IT_TXFT TXFIFO threshold interrupt
  – UART_IT_CM Character match interrupt
  – UART_IT_CTS CTS change interrupt
  – UART_IT_LBD LIN Break detection interrupt
  – UART_IT_TXE Transmit Data Register empty interrupt
  – UART_IT_TXFNF TX FIFO not full interrupt
  – UART_IT_TC Transmission complete interrupt
  – UART_IT_RXNE Receive Data register not empty interrupt
  – UART_IT_RXFNE RXFIFO not empty interrupt
  – UART_IT_RTO Receive Timeout interrupt
  – UART_IT_IDLE Idle line detection interrupt
  – UART_IT_PE Parity Error interrupt
  – UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:
- None

__HAL_UART_GET_IT

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_GET_IT_SOURCE

Description:
- Check whether the specified UART interrupt source is enabled or not.

Parameters:
- __HANDLE__: specifies the UART Handle.
- __INTERRUPT__: specifies the UART interrupt source to check. This parameter can be one of the following values:
  - UART_IT_RXFF RXFIFO Full interrupt
  - UART_IT_TXFE TXFIFO Empty interrupt
  - UART_IT_RXFT RXFIFO threshold interrupt
  - UART_IT_TXFT TXFIFO threshold interrupt
  - UART_IT_CM Character match interrupt
  - UART_IT_CTS CTS change interrupt
  - UART_IT_LBD LIN Break detection interrupt
  - UART_IT_TXE Transmit Data Register empty interrupt
  - UART_IT_TXFNF TX FIFO not full interrupt
  - UART_IT_TC Transmission complete interrupt
  - UART_IT_RXNE Receive Data register not empty interrupt
  - UART_IT_RXFNE RXFIFO not empty interrupt
  - UART_IT_RTO Receive Timeout interrupt
  - UART_IT_IDLE Idle line detection interrupt
  - UART_IT_PE Parity Error interrupt
  - UART_IT_ERR Error interrupt (Frame error, noise error, overrun error)

Return value:
- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_UART_CLEAR_IT

Description:
- Clear the specified UART ISR flag, in setting the proper ICR register flag.

Parameters:
- __HANDLE__: specifies the UART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt This parameter can be one of the following values:
  - UART_CLEAR_PEF Parity Error Clear Flag
  - UART_CLEAR_FEF Framing Error Clear Flag
  - UART_CLEAR_NEF Noise detected Clear Flag
  - UART_CLEAR_OREF Overrun Error Clear Flag
  - UART_CLEAR_IDLEF IDLE line detected Clear Flag
  - UART_CLEAR_RTOF Receiver timeout clear flag
  - UART_CLEAR_TXFECF TXFIFO empty Clear Flag
  - UART_CLEAR_TCF Transmission Complete Clear Flag
  - UART_CLEAR_LBDF LIN Break Detection Clear Flag
  - UART_CLEAR_CTSF CTS Interrupt Clear Flag
  - UART_CLEAR_CMF Character Match Clear Flag

Return value:
- None

__HAL_UART_SEND_REQ

Description:
- Set a specific UART request flag.

Parameters:
- __HANDLE__: specifies the UART Handle.
- __REQ__: specifies the request flag to set This parameter can be one of the following values:
  – UART_AUTOBAUD_REQUEST Auto-Baud Rate Request
  – UART_SENDBREAK_REQUEST Send Break Request
  – UART_MUTE_MODE_REQUEST Mute Mode Request
  – UART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  – UART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:
- None

__HAL_UART_ONE_BIT_SAMPLE_ENABLE

Description:
- Enable the UART one bit sample method.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_ONE_BIT_SAMPLE_DISABLE

Description:
- Disable the UART one bit sample method.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_ENABLE

Description:
- Enable UART.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_DISABLE

Description:
- Disable UART.

Parameters:
- __HANDLE__: specifies the UART Handle.

Return value:
- None

__HAL_UART_HWCONTROL_CTS_ENABLE

Description:

- Enable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_CTS_DISABLE

Description:

- Disable CTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable CTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_ENABLE

Description:

- Enable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to enable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

__HAL_UART_HWCONTROL_RTS_DISABLE

Description:

- Disable RTS flow control.

Parameters:

- __HANDLE__: specifies the UART Handle.

Return value:

- None

Notes:

- This macro allows to disable RTS hardware flow control for a given UART instance, without need to call HAL_UART_Init() function. As involving direct access to UART registers, usage of this macro should be fully endorsed by user. As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e. __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e. __HAL_UART_ENABLE(__HANDLE__)).

UART Status Flags

UART_FLAG_TXFT

UART TXFIFO threshold flag

UART_FLAG_RXFT

UART RXFIFO threshold flag

UART_FLAG_RXFF

UART RXFIFO Full flag

UART_FLAG_TXFE

UART TXFIFO Empty flag

UART_FLAG_REACK

UART receive enable acknowledge flag

UART_FLAG_TEACK

UART transmit enable acknowledge flag

UART_FLAG_RWU

UART receiver wake-up from mute mode flag

UART_FLAG_SBKF

UART send break flag

UART_FLAG_CMF

UART character match flag

UART_FLAG_BUSY

UART busy flag

UART_FLAG_ABRF

UART auto Baud rate flag

UART_FLAG_ABRE

UART auto Baud rate error

UART_FLAG_RTOF

UART receiver timeout flag

UART_FLAG_CTS

UART clear to send flag

UART_FLAG_CTSIF

UART clear to send interrupt flag

UART_FLAG_LBDF

UART LIN break detection flag

UART_FLAG_TXE

UART transmit data register empty

UART_FLAG_TXFNF

UART TXFIFO not full

UART_FLAG_TC

UART transmission complete

UART_FLAG_RXNE

UART read data register not empty

UART_FLAG_RXFNE

UART RXFIFO not empty

UART_FLAG_IDLE

UART idle flag

UART_FLAG_ORE

UART overrun error

UART_FLAG_NE

UART noise error

UART_FLAG_FE

UART frame error

UART_FLAG_PE

UART parity error

UART Half Duplex Selection

UART_HALF_DUPLEX_DISABLE

UART half-duplex disabled

UART_HALF_DUPLEX_ENABLE

UART half-duplex enabled

UART Hardware Flow Control

UART_HWCONTROL_NONE

No hardware control

UART_HWCONTROL_RTS

Request To Send

UART_HWCONTROL_CTS

Clear To Send

UART_HWCONTROL_RTS_CTS

Request and Clear To Send

UART Interruptions Flag Mask

UART_IT_MASK

UART interruptions flags mask

UART Interrupts Definition

UART_IT_PE

UART parity error interruption

UART_IT_TXE

UART transmit data register empty interruption

UART_IT_TXFNF

UART TX FIFO not full interruption

UART_IT_TC

UART transmission complete interruption

UART_IT_RXNE

UART read data register not empty interruption

UART_IT_RXFNE

UART RXFIFO not empty interruption

UART_IT_IDLE

UART idle interruption

UART_IT_LBD

UART LIN break detection interruption

UART_IT_CTS

UART CTS interruption

UART_IT_CM

UART character match interruption

UART_IT_RXFF

UART RXFIFO full interruption

UART_IT_TXFE

UART TXFIFO empty interruption

UART_IT_RXFT

UART RXFIFO threshold reached interruption

UART_IT_TXFT

UART TXFIFO threshold reached interruption

UART_IT_RTO

UART receiver timeout interruption

UART_IT_ERR

UART error interruption

UART_IT_ORE

UART overrun error interruption

UART_IT_NE

UART noise error interruption

UART_IT_FE

UART frame error interruption

UART Interruption Clear Flags

UART_CLEAR_PEF

Parity Error Clear Flag

UART_CLEAR_FEF

Framing Error Clear Flag

UART_CLEAR_NEF

Noise Error detected Clear Flag

UART_CLEAR_OREF

Overrun Error Clear Flag

UART_CLEAR_IDLEF

IDLE line detected Clear Flag

UART_CLEAR_TXFECF

TXFIFO empty clear flag

UART_CLEAR_TCF

Transmission Complete Clear Flag

UART_CLEAR_LBDF

LIN Break Detection Clear Flag

UART_CLEAR_CTSF

CTS Interrupt Clear Flag

UART_CLEAR_CMF

Character Match Clear Flag

UART_CLEAR_RTOF

UART receiver timeout clear flag

UART Local Interconnection Network mode

UART_LIN_DISABLE

Local Interconnect Network disable

UART_LIN_ENABLE

Local Interconnect Network enable

UART LIN Break Detection

UART_LINBREAKDETECTLENGTH_10B

LIN 10-bit break detection length

UART_LINBREAKDETECTLENGTH_11B

LIN 11-bit break detection length

UART Transfer Mode

UART_MODE_RX

RX mode

UART_MODE_TX

TX mode

UART_MODE_TX_RX

RX and TX mode

UART Advanced Feature MSB First

UART_ADVFEATURE_MSBFIRST_DISABLE

Most significant bit sent/received first disable

UART_ADVFEATURE_MSBFIRST_ENABLE

Most significant bit sent/received first enable

UART Advanced Feature Mute Mode Enable

UART_ADVFEATURE_MUTEMODE_DISABLE

UART mute mode disable

UART_ADVFEATURE_MUTEMODE_ENABLE

UART mute mode enable

UART One Bit Sampling Method

UART_ONE_BIT_SAMPLE_DISABLE

One-bit sampling disable

UART_ONE_BIT_SAMPLE_ENABLE

One-bit sampling enable

UART Advanced Feature Overrun Disable

UART_ADVFEATURE_OVERRUN_ENABLE

RX overrun enable

UART_ADVFEATURE_OVERRUN_DISABLE

RX overrun disable

UART Over Sampling

UART_OVERSAMPLING_16

Oversampling by 16

UART_OVERSAMPLING_8

Oversampling by 8

UART Parity

UART_PARITY_NONE

No parity

UART_PARITY_EVEN

Even parity

UART_PARITY_ODD

Odd parity

UART Receiver Timeout

UART_RECEIVER_TIMEOUT_DISABLE

UART Receiver Timeout disable

UART_RECEIVER_TIMEOUT_ENABLE

UART Receiver Timeout enable

UART Reception type values

HAL_UART_RECEPTION_STANDARD

Standard reception

HAL_UART_RECEPTION_TOIDLE

Reception till completion or IDLE event

HAL_UART_RECEPTION_TORTO

Reception till completion or RTO event

HAL_UART_RECEPTION_TOCHARMATCH

Reception till completion or CM event

UART Request Parameters

UART_AUTOBAUD_REQUEST

Auto-Baud Rate Request

UART_SENDBREAK_REQUEST

Send Break Request

UART_MUTE_MODE_REQUEST

Mute Mode Request

UART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

UART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

UART RxEvent type values

HAL_UART_RXEVENT_TC

RxEvent linked to Transfer Complete event

HAL_UART_RXEVENT_HT

RxEvent linked to Half Transfer event

HAL_UART_RXEVENT_IDLE

RxEvent linked to IDLE event

UART Advanced Feature RX Pin Active Level Inversion

UART_ADVFEATURE_RXINV_DISABLE

RX pin active level inversion disable

UART_ADVFEATURE_RXINV_ENABLE

RX pin active level inversion enable

UART Advanced Feature RX TX Pins Swap

UART_ADVFEATURE_SWAP_DISABLE

TX/RX pins swap disable

UART_ADVFEATURE_SWAP_ENABLE

TX/RX pins swap enable

UART State

UART_STATE_DISABLE

UART disabled

UART_STATE_ENABLE

UART enabled

UART State Code Definition

HAL_UART_STATE_RESET

Peripheral is not initialized Value is allowed for gState and RxState

HAL_UART_STATE_READY

Peripheral Initialized and ready for use Value is allowed for gState and RxState

HAL_UART_STATE_BUSY

an internal process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_TX

Data Transmission process is ongoing Value is allowed for gState only

HAL_UART_STATE_BUSY_RX

Data Reception process is ongoing Value is allowed for RxState only

HAL_UART_STATE_BUSY_TX_RX

Data Transmission and Reception process is ongoing Not to be used for neither gState nor RxState.Value is result of combination (Or) between gState and RxState values

HAL_UART_STATE_TIMEOUT

Timeout state Value is allowed for gState only

HAL_UART_STATE_ERROR

Error Value is allowed for gState only

UART Number of Stop Bits

UART_STOPBITS_0_5

UART frame with 0.5 stop bit

UART_STOPBITS_1

UART frame with 1 stop bit

UART_STOPBITS_1_5

UART frame with 1.5 stop bits

UART_STOPBITS_2

UART frame with 2 stop bits

UART Advanced Feature Stop Mode Enable

UART_ADVFEATURE_STOPMODE_DISABLE

UART stop mode disable

UART_ADVFEATURE_STOPMODE_ENABLE

UART stop mode enable

UART polling-based communications time-out value

HAL_UART_TIMEOUT_VALUE

UART polling-based communications time-out value

UART Advanced Feature TX Pin Active Level Inversion

UART_ADVFEATURE_TXINV_DISABLE

TX pin active level inversion disable

UART_ADVFEATURE_TXINV_ENABLE

TX pin active level inversion enable

UART WakeUp From Stop Selection

UART_WAKEUP_ON_ADDRESS

UART wake-up on address

UART_WAKEUP_ON_READDATA_NONEMPTY

UART wake-up on receive data register not empty or RXFIFO is not empty

UART WakeUp Methods

UART_WAKEUPMETHOD_IDLELINE

UART wake-up on idle line

UART_WAKEUPMETHOD_ADDRESSMARK

UART wake-up on address mark

85 HAL UART Extension Driver

85.1 UARTEx Firmware driver registers structures

85.1.1 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32u5xx_hal_uart_ex.h

Data Fields

- ***uint32_t WakeUpEvent***
- ***uint16_t AddressLength***
- ***uint8_t Address***

Field Documentation

- ***uint32_t UART_WakeUpTypeDef::WakeUpEvent***
  Specifies which event will activate the Wakeup from Stop mode flag (WUF). This parameter can be a value of ***UART_WakeUp_from_Stop_Selection***. If set to UART_WAKEUP_ON_ADDRESS, the two other fields below must be filled up.
- ***uint16_t UART_WakeUpTypeDef::AddressLength***
  Specifies whether the address is 4 or 7-bit long. This parameter can be a value of ***UARTEx_WakeUp_Address_Length***.
- ***uint8_t UART_WakeUpTypeDef::Address***
  UART/USART node address (7-bit long max).

85.1.2 UART_AutonomousModeConfTypeDef

UART_AutonomousModeConfTypeDef is defined in the stm32u5xx_hal_uart_ex.h

Data Fields

- ***uint32_t AutonomousModeState***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerPolarity***
- ***uint32_t DataSize***
- ***uint32_t IdleFrame***

Field Documentation

- ***uint32_t UART_AutonomousModeConfTypeDef::AutonomousModeState***
  Specifies the autonomous mode state.This parameter can be a value of ***UARTEx_Autonomous_mode***.
- ***uint32_t UART_AutonomousModeConfTypeDef::TriggerSelection***
  Specifies which trigger will activate the Transmission automatically. This parameter can be a value of ***UARTEx_Autonomous_Trigger_selection*** or ***UARTEx_Low_Power_Autonomous_Trigger_selection***.
- ***uint32_t UART_AutonomousModeConfTypeDef::TriggerPolarity***
  Specifies the autonomous mode trigger signal polarity. This parameter can be a value of ***UARTEx_Autonomous_Trigger_Polarity***
- ***uint32_t UART_AutonomousModeConfTypeDef::DataSize***
  Specifies the transmitted data size in byte
- ***uint32_t UART_AutonomousModeConfTypeDef::IdleFrame***
  Specifies whether the IDLE frame transmission is enabled or disabled. This parameter can be a value of ***UARTEx_Autonomous_IDLE_FRAME***.

85.2 UARTEx Firmware driver API description

The following section lists the various functions of the UARTEx library.

85.2.1 UART peripheral extended features

85.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode the parameters below can be configured:
  - Baud Rate
  - Word Length
  - Stop Bit
  - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  - Hardware flow control
  - Receiver/transmitter modes
  - Over Sampling Method
  - One-Bit Sampling Method
- For the asynchronous mode, the following advanced features can be configured as well:
  - TX and/or RX pin level inversion
  - data logical level inversion
  - RX and TX pins swap
  - RX overrun detection disabling
  - DMA disabling on RX error
  - MSB first on communication line
  - auto Baud rate detection

The HAL_RS485Ex_Init() API follows the UART RS485 mode configuration procedures (details for the procedures are available in reference manual).

This section contains the following APIs:

- ***HAL_RS485Ex_Init()***

85.2.3 IO operation functions

This section contains the following APIs:

- ***HAL_UARTEx_RxFifoFullCallback()***
- ***HAL_UARTEx_TxFifoEmptyCallback()***
- ***HAL_UARTEx_WakeupCallback()***

85.2.4 Peripheral Control functions

This section provides the following functions:

- HAL_MultiProcessorEx_AddressLength_Set() API optionally sets the UART node address detection length to more than 4 bits for multiprocessor address mark wake up.
- HAL_UARTEx_StopModeWakeUpSourceConfig() API defines the wake-up from stop mode trigger: address match, Start Bit detection or RXNE bit status.
- HAL_UARTEx_EnableStopMode() API enables the UART to wake up the MCU from stop mode
- HAL_UARTEx_DisableStopMode() API disables the above functionality
- HAL_UARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_UARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_UARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_UARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown).

1. Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller : (+) Detection of inactivity period (RX line has not been active for a given period).
   - RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
   - RX inactivity detected by RTO, i.e. line has been in idle state for a programmable time, after last received byte. (+) Detection that a specific character has been received.

2. There are two mode of transfer: (+) Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer. (+) Non-Blocking mode: The reception is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected.
3. Blocking mode API: (+) HAL_UARTEx_ReceiveToIdle()
4. Non-Blocking mode API with Interrupt: (+) HAL_UARTEx_ReceiveToIdle_IT()
5. Non-Blocking mode API with DMA: (+) HAL_UARTEx_ReceiveToIdle_DMA()

This subsection also provides a set of additional functions providing enhanced reception services to user. (For example, these functions allow application to handle use cases where number of data to be received is unknown). (#) Compared to standard reception services which only consider number of received data elements as reception completion criteria, these functions also consider additional events as triggers for updating reception status to caller :

- Detection of inactivity period (RX line has not been active for a given period).
  – RX inactivity detected by IDLE event, i.e. RX line has been in idle state (normally high state) for 1 frame time, after last received byte.
  – RX inactivity detected by RTO, i.e. line has been in idle state for a programmable time, after last received byte.
- Detection that a specific character has been received. (#) There are two mode of transfer:
- Blocking mode: The reception is performed in polling mode, until either expected number of data is received, or till IDLE event occurs. Reception is handled only during function execution. When function exits, no data reception could occur. HAL status and number of actually received data elements, are returned by function after finishing transfer.
- Non-Blocking mode: The reception is performed using Interrupts or DMA. These API's return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UARTEx_RxEventCallback() user callback will be executed during Receive process The HAL_UART_ErrorCallback()user callback will be executed when a reception error is detected. (#) Blocking mode API:
- HAL_UARTEx_ReceiveToIdle() (#) Non-Blocking mode API with Interrupt:
- HAL_UARTEx_ReceiveToIdle_IT() (#) Non-Blocking mode API with DMA:
- HAL_UARTEx_ReceiveToIdle_DMA()

This section contains the following APIs:

- ***HAL_MultiProcessorEx_AddressLength_Set()***
- ***HAL_UARTEx_StopModeWakeUpSourceConfig()***
- ***HAL_UARTEx_EnableStopMode()***
- ***HAL_UARTEx_DisableStopMode()***
- ***HAL_UARTEx_EnableFifoMode()***
- ***HAL_UARTEx_DisableFifoMode()***
- ***HAL_UARTEx_SetTxFifoThreshold()***
- ***HAL_UARTEx_SetRxFifoThreshold()***
- ***HAL_UARTEx_ReceiveToIdle()***
- ***HAL_UARTEx_ReceiveToIdle_IT()***
- ***HAL_UARTEx_ReceiveToIdle_DMA()***
- ***HAL_UARTEx_GetRxEventType()***
- ***HAL_UARTEx_SetConfigAutonomousMode()***
- ***HAL_UARTEx_GetConfigAutonomousMode()***
- ***HAL_UARTEx_ClearConfigAutonomousMode()***

85.2.5 Detailed description of functions

HAL_RS485Ex_Init

Function name

HAL_StatusTypeDef HAL_RS485Ex_Init (UART_HandleTypeDef * huart, uint32_t Polarity, uint32_t AssertionTime, uint32_t DeassertionTime)

Function description

Initialize the RS485 Driver enable feature according to the specified parameters in the UART_InitTypeDef and creates the associated handle.

Parameters

- **huart:** UART handle.
- **Polarity:** Select the driver enable polarity. This parameter can be one of the following values:
  – UART_DE_POLARITY_HIGH DE signal is active high
  – UART_DE_POLARITY_LOW DE signal is active low
- **AssertionTime:** Driver Enable assertion time: 5-bit value defining the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate)
- **DeassertionTime:** Driver Enable deassertion time: 5-bit value defining the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit time, depending on the oversampling rate).

Return values

- **HAL:** status

HAL_UARTEx_WakeupCallback

Function name

void HAL_UARTEx_WakeupCallback (UART_HandleTypeDef * huart)

Function description

HAL_UARTEx_RxFifoFullCallback

Function name

void HAL_UARTEx_RxFifoFullCallback (UART_HandleTypeDef * huart)

Function description

UART RX Fifo full callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_TxFifoEmptyCallback

Function name

void HAL_UARTEx_TxFifoEmptyCallback (UART_HandleTypeDef * huart)

Function description

UART TX Fifo empty callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

HAL_UARTEx_StopModeWakeUpSourceConfig

Function name

HAL_StatusTypeDef HAL_UARTEx_StopModeWakeUpSourceConfig (UART_HandleTypeDef * huart, UART_WakeUpTypeDef WakeUpSelection)

Function description

Set Wakeup from Stop mode interrupt flag selection.

Parameters

- **huart:** UART handle.
- **WakeUpSelection:** Address match, Start Bit detection or RXNE/RXFNE bit status. This parameter can be one of the following values:
  – UART_WAKEUP_ON_ADDRESS
  – UART_WAKEUP_ON_READDATA_NONEMPTY

Return values

- **HAL:** status

Notes

- It is the application responsibility to enable the interrupt used as usart_wkup interrupt source before entering low-power mode.

HAL_UARTEx_EnableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableStopMode (UART_HandleTypeDef * huart)

Function description

Enable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

Notes

- The UART is able to wake up the MCU from Stop 1 mode as long as UART clock is HSI or LSE.

HAL_UARTEx_DisableStopMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableStopMode (UART_HandleTypeDef * huart)

Function description

Disable UART Stop Mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessorEx_AddressLength_Set

Function name

HAL_StatusTypeDef HAL_MultiProcessorEx_AddressLength_Set (UART_HandleTypeDef * huart, uint32_t AddressLength)

Function description

By default in multiprocessor mode, when the wake up method is set to address mark, the UART handles only 4-bit long addresses detection; this API allows to enable longer addresses detection (6-, 7- or 8-bit long).

Parameters

- **huart:** UART handle.
- **AddressLength:** This parameter can be one of the following values:
  - UART_ADDRESS_DETECT_4B 4-bit long address
  - UART_ADDRESS_DETECT_7B 6-, 7- or 8-bit long address

Return values

- **HAL:** status

Notes

- Addresses detection lengths are: 6-bit address detection in 7-bit data mode, 7-bit address detection in 8-bit data mode, 8-bit address detection in 9-bit data mode.

HAL_UARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_EnableFifoMode (UART_HandleTypeDef * huart)

Function description

Enable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_UARTEx_DisableFifoMode (UART_HandleTypeDef * huart)

Function description

Disable the FIFO mode.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_UARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetTxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - UART_TXFIFO_THRESHOLD_1_8
  - UART_TXFIFO_THRESHOLD_1_4
  - UART_TXFIFO_THRESHOLD_1_2
  - UART_TXFIFO_THRESHOLD_3_4
  - UART_TXFIFO_THRESHOLD_7_8
  - UART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_UARTEx_SetRxFifoThreshold (UART_HandleTypeDef * huart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **huart:** UART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  - UART_RXFIFO_THRESHOLD_1_8
  - UART_RXFIFO_THRESHOLD_1_4
  - UART_RXFIFO_THRESHOLD_1_2
  - UART_RXFIFO_THRESHOLD_3_4
  - UART_RXFIFO_THRESHOLD_7_8
  - UART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_UARTEx_ReceiveToIdle

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint16_t * RxLen, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.
- **RxLen:** Number of data elements finally received (could be lower than Size, in case reception ends on IDLE event)
- **Timeout:** Timeout duration expressed in ms (covers the whole reception sequence).

Return values

- **HAL:** status

Notes

- HAL_OK is returned if reception is completed (expected number of data has been received) or if reception is stopped after IDLE event (less than the expected number of data has been received) In this case, RxLen output parameter indicates number of data available in reception buffer.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.
- When FIFO mode is enabled, the RXFNE flag is set as long as the RXFIFO is not empty. Read operations from the RDR register are performed when RXFNE flag is set. From hardware perspective, RXFNE flag and RXNE are mapped on the same bit-field.

HAL_UARTEx_ReceiveToIdle_IT

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.

Return values

- **HAL:** status

Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to UART interrupts raised by RXNE and IDLE events. Callback is called at end of reception indicating number of received data elements.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.

HAL_UARTEx_ReceiveToIdle_DMA

Function name

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Receive an amount of data in DMA mode till either the expected number of data is received or an IDLE event occurs.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (uint8_t or uint16_t data elements).
- **Size:** Amount of data elements (uint8_t or uint16_t) to be received.

Return values

- **HAL:** status

Notes

- Reception is initiated by this function call. Further progress of reception is achieved thanks to DMA services, transferring automatically received data elements in user reception buffer and calling registered callbacks at half/end of reception. UART IDLE events are also used to consider reception phase as ended. In all cases, callback execution will indicate number of received data elements.
- When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of uint16_t. In this case, Size must indicate the number of uint16_t available through pData.

HAL_UARTEx_GetRxEventType

Function name

HAL_UART_RxEventTypeTypeDef HAL_UARTEx_GetRxEventType (const UART_HandleTypeDef * huart)

Function description

Provide Rx Event type that has lead to RxEvent callback execution.

Parameters

- **huart:** UART handle.

Return values

- **Rx:** Event Type (return vale will be a value of UART RxEvent type values)

Notes

- When HAL_UARTEx_ReceiveToIdle_IT() or HAL_UARTEx_ReceiveToIdle_DMA() API are called, progress of reception process is provided to application through calls of Rx Event callback (either default one HAL_UARTEx_RxEventCallback() or user registered one). As several types of events could occur (IDLE event, Half Transfer, or Transfer Complete), this function allows to retrieve the Rx Event type that has lead to Rx Event callback execution.
- This function is expected to be called within the user implementation of Rx Event Callback, in order to provide the accurate value : In Interrupt Mode : HAL_UART_RXEVENT_TC : when Reception has been completed (expected nb of data has been received)HAL_UART_RXEVENT_IDLE : when Idle event occurred prior reception has been completed (nb of received data is lower than expected one) In DMA Mode :HAL_UART_RXEVENT_TC : when Reception has been completed (expected nb of data has been received)HAL_UART_RXEVENT_HT : when half of expected nb of data has been receivedHAL_UART_RXEVENT_IDLE : when Idle event occurred prior reception has been completed (nb of received data is lower than expected one). In DMA mode, RxEvent callback could be called several times; When DMA is configured in Normal Mode, HT event does not stop Reception process; When DMA is configured in Circular Mode, HT, TC or IDLE events don't stop Reception process;

HAL_UARTEx_SetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_UARTEx_SetConfigAutonomousMode (UART_HandleTypeDef * huart, const UART_AutonomousModeConfTypeDef * sConfig)

Function description

Set autonomous mode Configuration.

Parameters

- **huart:** UART handle.
- **sConfig:** Autonomous mode structure parameters.

Return values

- **HAL:** status

HAL_UARTEx_GetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_UARTEx_GetConfigAutonomousMode (const UART_HandleTypeDef * huart, UART_AutonomousModeConfTypeDef * sConfig)

Function description

Get autonomous mode Configuration.

Parameters

- **huart:** UART handle.
- **sConfig:** Autonomous mode structure parameters.

Return values

- **HAL:** status

HAL_UARTEx_ClearConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_UARTEx_ClearConfigAutonomousMode (UART_HandleTypeDef * huart)

Function description

Clear autonomous mode Configuration.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

85.3 UARTEx Firmware driver defines

The following section lists the various define and macros of the module.

85.3.1 UARTEx

UARTEx

UARTEx Autonomous IDLE Frame

UART_IDLE_FRAME_ENABLE

IDLE Frame sent after enabling the transmitter

UART_IDLE_FRAME_DISABLE

IDLE Frame not sent after enabling the transmitter

UARTEx Autonomous Mode

UART_AUTONOMOUS_MODE_DISABLE

Autonomous mode disable

UART_AUTONOMOUS_MODE_ENABLE

Autonomous mode enable

UARTEx Autonomous Trigger Polarity

UART_TRIG_POLARITY_RISING

UART triggered on rising edge

UART_TRIG_POLARITY_FALLING

UART triggered on falling edge

UARTEx Autonomous trigger selection

UART_GPDMA1_CH0_TCF_TRG

UART GPDMA1 channel0 Internal Trigger

UART_GPDMA1_CH1_TCF_TRG

UART GPDMA1 channel1 Internal Trigger

UART_GPDMA1_CH2_TCF_TRG

UART GPDMA1 channel2 Internal Trigger

UART_GPDMA1_CH3_TCF_TRG

UART GPDMA1 channel3 Internal Trigger

UART_EXTI_LINE6_TRG

UART EXTI line 6 Internal Trigger

UART_EXTI_LINE9_TRG

UART EXTI line 9 Internal Trigger

UART_LPTIM1_OUT_TRG

UART LPTIM1 out Internal Trigger

UART_LPTIM2_OUT_TRG

UART LPTIM2 out Internal Trigger

UART_COMP1_OUT_TRG

UART COMP1 out Internal Trigger

UART_COMP2_OUT_TRG

UART COMP2 out Internal Trigger

UART_RTC_ALRA_TRG

UART RTC alarm Internal Trigger

UART_RTC_WUT_TRG

UART RTC wakeup Internal Trigger

UARTEx FIFO mode

UART_FIFOMODE_DISABLE

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

UARTEx Low Power Autonomous trigger selection

LPUART_LPDMA1_CH0_TCF_TRG

LPUART LPDMA1 channel0 Internal Trigger

LPUART_LPDMA1_CH1_TCF_TRG

LPUART LPDMA1 channel1 Internal Trigger

LPUART_LPDMA1_CH2_TCF_TRG
LPUART LPDMA1 channel2 Internal Trigger

LPUART_LPDMA1_CH3_TCF_TRG
LPUART LPDMA1 channel3 Internal Trigger

LPUART_EXTI_LINE6_TRG
LPUART EXTI line 6 Internal Trigger

LPUART_EXTI_LINE8_TRG
LPUART EXTI line 8 Internal Trigger

LPUART_LPTIM1_OUT_TRG
LPUART LPTIM1 out Internal Trigger

LPUART_LPTIM3_OUT_TRG
LPUART LPTIM3 out Internal Trigger

LPUART_COMP1_OUT_TRG
LPUART COMP1 out Internal Trigger

LPUART_COMP2_OUT_TRG
LPUART COMP2 out Internal Trigger

LPUART_RTC_ALRA_TRG
LPUART RTC alarm Internal Trigger

LPUART_RTC_WUT_TRG
LPUART RTC wakeup Internal Trigger

UARTEx RXFIFO threshold level

UART_RXFIFO_THRESHOLD_1_8
RX FIFO reaches 1/8 of its depth

UART_RXFIFO_THRESHOLD_1_4
RX FIFO reaches 1/4 of its depth

UART_RXFIFO_THRESHOLD_1_2
RX FIFO reaches 1/2 of its depth

UART_RXFIFO_THRESHOLD_3_4
RX FIFO reaches 3/4 of its depth

UART_RXFIFO_THRESHOLD_7_8
RX FIFO reaches 7/8 of its depth

UART_RXFIFO_THRESHOLD_8_8
RX FIFO becomes full

UARTEx TXFIFO threshold level

UART_TXFIFO_THRESHOLD_1_8
TX FIFO reaches 1/8 of its depth

UART_TXFIFO_THRESHOLD_1_4
TX FIFO reaches 1/4 of its depth

UART_TXFIFO_THRESHOLD_1_2

TX FIFO reaches 1/2 of its depth

UART_TXFIFO_THRESHOLD_3_4

TX FIFO reaches 3/4 of its depth

UART_TXFIFO_THRESHOLD_7_8

TX FIFO reaches 7/8 of its depth

UART_TXFIFO_THRESHOLD_8_8

TX FIFO becomes empty

UARTEx WakeUp Address Length

UART_ADDRESS_DETECT_4B

4-bit long wake-up address

UART_ADDRESS_DETECT_7B

7-bit long wake-up address

UARTEx Word Length

UART_WORDLENGTH_7B

7-bit long UART frame

UART_WORDLENGTH_8B

8-bit long UART frame

UART_WORDLENGTH_9B

9-bit long UART frame

86 HAL USART Generic Driver

86.1 USART Firmware driver registers structures

86.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32u5xx_hal_usart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t CLKPolarity***
- ***uint32_t CLKPhase***
- ***uint32_t CLKLastBit***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***
  This member configures the Usart communication baud rate. The baud rate is computed using the following formula: Baud Rate Register[15:4] = ((2 * fclk_pres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * fclk_pres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where fclk_pres is the USART input clock frequency (fclk) divided by a prescaler.
  Note:
  – Oversampling by 8 is systematically applied to achieve high baud rates.
- ***uint32_t USART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USARTEx_Word_Length***.
- ***uint32_t USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_Stop_Bits***.
- ***uint32_t USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_Mode***.
- ***uint32_t USART_InitTypeDef::CLKPolarity***
  Specifies the steady state of the serial clock. This parameter can be a value of ***USART_Clock_Polarity***.
- ***uint32_t USART_InitTypeDef::CLKPhase***
  Specifies the clock transition on which the bit capture is made. This parameter can be a value of ***USART_Clock_Phase***.
- ***uint32_t USART_InitTypeDef::CLKLastBit***
  Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of ***USART_Last_Bit***.
- ***uint32_t USART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the USART clock source. This parameter can be a value of ***USART_ClockPrescaler***.

86.1.2 __USART_HandleTypeDef

__USART_HandleTypeDef is defined in the stm32u5xx_hal_usart.h

Data Fields

- *USART_TypeDef * Instance*
- *USART_InitTypeDef Init*
- *const uint8_t * pTxBuffPtr*
- *uint16_t TxXferSize*
- *__IO uint16_t TxXferCount*
- *uint8_t * pRxBuffPtr*
- *uint16_t RxXferSize*
- *__IO uint16_t RxXferCount*
- *uint16_t Mask*
- *uint16_t NbRxDataToProcess*
- *uint16_t NbTxDataToProcess*
- *uint32_t SlaveMode*
- *uint32_t FifoMode*
- *void(* RxISR*
- *void(* TxISR*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_USART_StateTypeDef State*
- *__IO uint32_t ErrorCode*
- *void(* TxHalfCpltCallback*
- *void(* TxCpltCallback*
- *void(* RxHalfCpltCallback*
- *void(* RxCpltCallback*
- *void(* TxRxCpltCallback*
- *void(* ErrorCallback*
- *void(* AbortCpltCallback*
- *void(* RxFifoFullCallback*
- *void(* TxFifoEmptyCallback*
- *void(* MspInitCallback*
- *void(* MspDeInitCallback*

Field Documentation

- ***USART_TypeDef* __USART_HandleTypeDef::Instance***
  USART registers base address
- ***USART_InitTypeDef __USART_HandleTypeDef::Init***
  USART communication parameters
- ***const uint8_t* __USART_HandleTypeDef::pTxBuffPtr***
  Pointer to USART Tx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::TxXferSize***
  USART Tx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::TxXferCount***
  USART Tx Transfer Counter
- ***uint8_t* __USART_HandleTypeDef::pRxBuffPtr***
  Pointer to USART Rx transfer Buffer
- ***uint16_t __USART_HandleTypeDef::RxXferSize***
  USART Rx Transfer size
- ***__IO uint16_t __USART_HandleTypeDef::RxXferCount***
  USART Rx Transfer Counter
- ***uint16_t __USART_HandleTypeDef::Mask***
  USART Rx RDR register mask
- ***uint16_t __USART_HandleTypeDef::NbRxDataToProcess***
  Number of data to process during RX ISR execution

- ***uint16_t __USART_HandleTypeDef::NbTxDataToProcess***
  Number of data to process during TX ISR execution
- ***uint32_t __USART_HandleTypeDef::SlaveMode***
  Enable/Disable UART SPI Slave Mode. This parameter can be a value of ***USARTEx_Slave_Mode***
- ***uint32_t __USART_HandleTypeDef::FifoMode***
  Specifies if the FIFO mode will be used. This parameter can be a value of ***USARTEx_FIFO_mode***.
- ***void(* __USART_HandleTypeDef::RxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Rx IRQ handler
- ***void(* __USART_HandleTypeDef::TxISR)(struct __USART_HandleTypeDef *husart)***
  Function pointer on Tx IRQ handler
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmatx***
  USART Tx DMA Handle parameters
- ***DMA_HandleTypeDef* __USART_HandleTypeDef::hdmarx***
  USART Rx DMA Handle parameters
- ***HAL_LockTypeDef __USART_HandleTypeDef::Lock***
  Locking object
- ***__IO HAL_USART_StateTypeDef __USART_HandleTypeDef::State***
  USART communication state
- ***__IO uint32_t __USART_HandleTypeDef::ErrorCode***
  USART Error code
- ***void(* __USART_HandleTypeDef::TxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Half Complete Callback
- ***void(* __USART_HandleTypeDef::TxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Complete Callback
- ***void(* __USART_HandleTypeDef::RxHalfCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Half Complete Callback
- ***void(* __USART_HandleTypeDef::RxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Complete Callback
- ***void(* __USART_HandleTypeDef::TxRxCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Rx Complete Callback
- ***void(* __USART_HandleTypeDef::ErrorCallback)(struct __USART_HandleTypeDef *husart)***
  USART Error Callback
- ***void(* __USART_HandleTypeDef::AbortCpltCallback)(struct __USART_HandleTypeDef *husart)***
  USART Abort Complete Callback
- ***void(* __USART_HandleTypeDef::RxFifoFullCallback)(struct __USART_HandleTypeDef *husart)***
  USART Rx Fifo Full Callback
- ***void(* __USART_HandleTypeDef::TxFifoEmptyCallback)(struct __USART_HandleTypeDef *husart)***
  USART Tx Fifo Empty Callback
- ***void(* __USART_HandleTypeDef::MspInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp Init callback
- ***void(* __USART_HandleTypeDef::MspDeInitCallback)(struct __USART_HandleTypeDef *husart)***
  USART Msp DeInit callback

86.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

86.2.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure (eg. USART_HandleTypeDef husart).

2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
   – Enable the USARTx interface clock.
   – USART pins configuration:
     ◦ Enable the clock for the USART GPIOs.
     ◦ Configure these USART pins as alternate function pull-up.
   – NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
     ◦ Configure the USARTx interrupt priority.
     ◦ Enable the NVIC USART IRQ handle.
   – USART interrupts handling:

Note: *The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.*

   – DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
     ◦ Declare a DMA handle structure for the Tx/Rx channel.
     ◦ Enable the DMAx interface clock.
     ◦ Configure the declared DMA handle structure with the required Tx/Rx parameters.
     ◦ Configure the DMA Tx/Rx channel.
     ◦ Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
     ◦ Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
3. Program the Baud Rate, Word Length, Stop Bit, Parity, and Mode (Receiver/Transmitter) in the husart handle Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
   – This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API.

Note: *To configure and enable/disable the USART to wake up the MCU from stop mode, resort to UART API's HAL_UARTEx_StopModeWakeUpSourceConfig(), HAL_UARTEx_EnableStopMode() and HAL_UARTEx_DisableStopMode() in casting the USART handle to UART type UART_HandleTypeDef.*

86.2.2 Callback registration

The compilation define USE_HAL_USART_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use Function HAL_USART_RegisterCallback() to register a user callback. Function HAL_USART_RegisterCallback() allows to register following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.
- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_USART_UnRegisterCallback() to reset a callback to the default weak function. HAL_USART_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:

- TxHalfCpltCallback : Tx Half Complete Callback.

- TxCpltCallback : Tx Complete Callback.
- RxHalfCpltCallback : Rx Half Complete Callback.
- RxCpltCallback : Rx Complete Callback.
- TxRxCpltCallback : Tx Rx Complete Callback.
- ErrorCallback : Error Callback.
- AbortCpltCallback : Abort Complete Callback.
- RxFifoFullCallback : Rx Fifo Full Callback.
- TxFifoEmptyCallback : Tx Fifo Empty Callback.
- MspInitCallback : USART MspInit.
- MspDeInitCallback : USART MspDeInit.

By default, after the HAL_USART_Init() and when the state is HAL_USART_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_USART_TxCpltCallback(), HAL_USART_RxHalfCpltCallback(). Exception done for MspInit and MspDeInit functions that are respectively reset to the legacy weak functions in the HAL_USART_Init() and HAL_USART_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_USART_Init() and HAL_USART_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand).

Callbacks can be registered/unregistered in HAL_USART_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_USART_STATE_READY or HAL_USART_STATE_RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_USART_RegisterCallback() before calling HAL_USART_DeInit() or HAL_USART_Init() function.

When The compilation define USE_HAL_USART_REGISTER_CALLBACKS is set to 0 or not defined, the callback registration feature is not available and weak callbacks are used.

86.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
  – Baud Rate
  – Word Length
  – Stop Bit
  – Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit.
  – USART polarity
  – USART phase
  – USART LastBit
  – Receiver/transmitter modes

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual).

This section contains the following APIs:

- *HAL_USART_Init()*
- *HAL_USART_DeInit()*
- *HAL_USART_MspInit()*
- *HAL_USART_MspDeInit()*
- *HAL_USART_RegisterCallback()*
- *HAL_USART_UnRegisterCallback()*

86.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
   – Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
   – No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_USART_ErrorCallback()user callback will be executed when a communication error is detected
2. Blocking mode API's are :
   – HAL_USART_Transmit() in simplex mode
   – HAL_USART_Receive() in full duplex receive only
   – HAL_USART_TransmitReceive() in full duplex mode
3. Non-Blocking mode API's with Interrupt are :
   – HAL_USART_Transmit_IT() in simplex mode
   – HAL_USART_Receive_IT() in full duplex receive only
   – HAL_USART_TransmitReceive_IT() in full duplex mode
   – HAL_USART_IRQHandler()
4. No-Blocking mode API's with DMA are :
   – HAL_USART_Transmit_DMA() in simplex mode
   – HAL_USART_Receive_DMA() in full duplex receive only
   – HAL_USART_TransmitReceive_DMA() in full duplex mode
   – HAL_USART_DMAPause()
   – HAL_USART_DMAResume()
   – HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in Non_Blocking mode:
   – HAL_USART_TxCpltCallback()
   – HAL_USART_RxCpltCallback()
   – HAL_USART_TxHalfCpltCallback()
   – HAL_USART_RxHalfCpltCallback()
   – HAL_USART_ErrorCallback()
   – HAL_USART_TxRxCpltCallback()
6. Non-Blocking mode transfers could be aborted using Abort API's :
   – HAL_USART_Abort()
   – HAL_USART_Abort_IT()
7. For Abort services based on interrupts (HAL_USART_Abort_IT), a Abort Complete Callbacks is provided:
   – HAL_USART_AbortCpltCallback()
8. In Non-Blocking mode transfers, possible errors are split into 2 categories. Errors are handled as follows :
   – Error is considered as Recoverable and non blocking : Transfer could go till end, but error severity is to be evaluated by user : this concerns Frame Error, Parity Error or Noise Error in Interrupt mode reception . Received character is then retrieved and stored in Rx buffer, Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed. Transfer is kept ongoing on USART side. If user wants to abort it, Abort services should be called by user.
   – Error is considered as Blocking : Transfer could not be completed properly and is aborted. This concerns Overrun Error In Interrupt mode reception and all errors in DMA mode. Error code is set to allow user to identify error type, and HAL_USART_ErrorCallback() user callback is executed.

This section contains the following APIs:
- *HAL_USART_Transmit()*
- *HAL_USART_Receive()*
- *HAL_USART_TransmitReceive()*
- *HAL_USART_Transmit_IT()*
- *HAL_USART_Receive_IT()*

- *HAL_USART_TransmitReceive_IT()*
- *HAL_USART_Transmit_DMA()*
- *HAL_USART_Receive_DMA()*
- *HAL_USART_TransmitReceive_DMA()*
- *HAL_USART_DMAPause()*
- *HAL_USART_DMAResume()*
- *HAL_USART_DMAStop()*
- *HAL_USART_Abort()*
- *HAL_USART_Abort_IT()*
- *HAL_USART_IRQHandler()*
- *HAL_USART_TxCpltCallback()*
- *HAL_USART_TxHalfCpltCallback()*
- *HAL_USART_RxCpltCallback()*
- *HAL_USART_RxHalfCpltCallback()*
- *HAL_USART_TxRxCpltCallback()*
- *HAL_USART_ErrorCallback()*
- *HAL_USART_AbortCpltCallback()*

86.2.5 Peripheral State and Error functions

This subsection provides functions allowing to :
- Return the USART handle state
- Return the USART handle error code

This section contains the following APIs:
- *HAL_USART_GetState()*
- *HAL_USART_GetError()*

86.2.6 Detailed description of functions

HAL_USART_Init

Function name

HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)

Function description

Initialize the USART mode according to the specified parameters in the USART_InitTypeDef and initialize the associated handle.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DeInit

Function name

HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART peripheral.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_MspInit

Function name

void HAL_USART_MspInit (USART_HandleTypeDef * husart)

Function description

Initialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_MspDeInit

Function name

void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)

Function description

DeInitialize the USART MSP.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_RegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID, pUSART_CallbackTypeDef pCallback)

Function description

Register a User USART Callback To be used to override the weak predefined callback.

Parameters

- **husart:** usart handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_USART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_USART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_USART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_USART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_TX_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_ERROR_CB_ID Error Callback ID
  - HAL_USART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_USART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_USART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_USART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_USART_MSPDEINIT_CB_ID MspDeInit Callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status +

Notes

- The HAL_USART_RegisterCallback() may be called before HAL_USART_Init() in HAL_USART_STATE_RESET to register callbacks for HAL_USART_MSPINIT_CB_ID and HAL_USART_MSPDEINIT_CB_ID

HAL_USART_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_USART_UnRegisterCallback (USART_HandleTypeDef * husart, HAL_USART_CallbackIDTypeDef CallbackID)

Function description

Unregister an USART Callback USART callaback is redirected to the weak predefined callback.

Parameters

- **husart:** usart handle
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_USART_TX_HALFCOMPLETE_CB_ID Tx Half Complete Callback ID
  - HAL_USART_TX_COMPLETE_CB_ID Tx Complete Callback ID
  - HAL_USART_RX_HALFCOMPLETE_CB_ID Rx Half Complete Callback ID
  - HAL_USART_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_TX_RX_COMPLETE_CB_ID Rx Complete Callback ID
  - HAL_USART_ERROR_CB_ID Error Callback ID
  - HAL_USART_ABORT_COMPLETE_CB_ID Abort Complete Callback ID
  - HAL_USART_RX_FIFO_FULL_CB_ID Rx Fifo Full Callback ID
  - HAL_USART_TX_FIFO_EMPTY_CB_ID Tx Fifo Empty Callback ID
  - HAL_USART_MSPINIT_CB_ID MspInit Callback ID
  - HAL_USART_MSPDEINIT_CB_ID MspDeInit Callback ID

Return values

- **HAL:** status

Notes

- The HAL_USART_UnRegisterCallback() may be called before HAL_USART_Init() in HAL_USART_STATE_RESET to un-register callbacks for HAL_USART_MSPINIT_CB_ID and HAL_USART_MSPDEINIT_CB_ID

HAL_USART_Transmit

Function name

HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size, uint32_t Timeout)

Function description

Simplex send an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be sent.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive

Function name

HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pRxData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

Function description

Full-Duplex Send and Receive an amount of data in blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).
- **Timeout:** Timeout duration.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_IT

Function name

HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_IT

Function name

HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- To receive synchronous data, dummy data are simultaneously transmitted.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_IT

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Send and Receive an amount of data in interrupt mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent (same amount to be received).

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint16_t Size)

Function description

Send an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be sent.

Return values

- **HAL:** status

Notes

- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data is handled as a set of u16. In this case, Size must indicate the number of u16 provided through pTxData.

HAL_USART_Receive_DMA

Function name

HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)

Function description

Receive an amount of data in DMA mode.

Parameters

- **husart:** USART handle.
- **pRxData:** pointer to data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position).
- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the received data is handled as a set of u16. In this case, Size must indicate the number of u16 available through pRxData.

HAL_USART_TransmitReceive_DMA

Function name

HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, const uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)

Function description

Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters

- **husart:** USART handle.
- **pTxData:** pointer to TX data buffer (u8 or u16 data elements).
- **pRxData:** pointer to RX data buffer (u8 or u16 data elements).
- **Size:** amount of data elements (u8 or u16) to be received/sent.

Return values

- **HAL:** status

Notes

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), the sent data and the received data are handled as sets of u16. In this case, Size must indicate the number of u16 available through pTxData and through pRxData.

HAL_USART_DMAPause

Function name

HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart)

Function description

Pause the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAResume

Function name

HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)

Function description

Resume the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_DMAStop

Function name

HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)

Function description

Stop the DMA Transfer.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USART_Abort

Function name

HAL_StatusTypeDef HAL_USART_Abort (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (blocking mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort (in case of transfer in DMA mode)Set handle State to READY
- This procedure is executed in blocking mode : when exiting function, Abort is considered as completed.

HAL_USART_Abort_IT

Function name

HAL_StatusTypeDef HAL_USART_Abort_IT (USART_HandleTypeDef * husart)

Function description

Abort ongoing transfers (Interrupt mode).

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- This procedure could be used for aborting any ongoing transfer started in Interrupt or DMA mode. This procedure performs following operations : Disable USART Interrupts (Tx and Rx)Disable the DMA transfer in the peripheral register (if enabled)Abort DMA transfer by calling HAL_DMA_Abort_IT (in case of transfer in DMA mode)Set handle State to READYAt abort completion, call user abort complete callback
- This procedure is executed in Interrupt mode, meaning that abort procedure could be considered as completed only when user abort complete callback is executed (not when exiting function).

HAL_USART_IRQHandler

Function name

void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)

Function description

Handle USART interrupt request.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxHalfCpltCallback

Function name

void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxCpltCallback

Function name

void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxCpltCallback

Function name

void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_RxHalfCpltCallback

Function name

void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)

Function description

Rx Half Transfer completed callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_TxRxCpltCallback

Function name

void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)

Function description

Tx/Rx Transfers completed callback for the non-blocking process.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_ErrorCallback

Function name

void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)

Function description

USART error callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_AbortCpltCallback

Function name

void HAL_USART_AbortCpltCallback (USART_HandleTypeDef * husart)

Function description

USART Abort Complete callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USART_GetState

Function name

HAL_USART_StateTypeDef HAL_USART_GetState (const USART_HandleTypeDef * husart)

Function description

Return the USART handle state.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle state

HAL_USART_GetError

Function name

uint32_t HAL_USART_GetError (const USART_HandleTypeDef * husart)

Function description

Return the USART error code.

Parameters

- **husart:** pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.

Return values

- **USART:** handle Error Code

86.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

86.3.1 USART

USART

USART Clock

USART_CLOCK_DISABLE

USART clock disable

USART_CLOCK_ENABLE

USART clock enable

USART Clock Prescaler

USART_PRESCALER_DIV1

fclk_pres = fclk

USART_PRESCALER_DIV2

fclk_pres = fclk/2

USART_PRESCALER_DIV4

fclk_pres = fclk/4

USART_PRESCALER_DIV6
fclk_pres = fclk/6

USART_PRESCALER_DIV8
fclk_pres = fclk/8

USART_PRESCALER_DIV10
fclk_pres = fclk/10

USART_PRESCALER_DIV12
fclk_pres = fclk/12

USART_PRESCALER_DIV16
fclk_pres = fclk/16

USART_PRESCALER_DIV32
fclk_pres = fclk/32

USART_PRESCALER_DIV64
fclk_pres = fclk/64

USART_PRESCALER_DIV128
fclk_pres = fclk/128

USART_PRESCALER_DIV256
fclk_pres = fclk/256

USART Clock Phase

USART_PHASE_1EDGE
USART frame phase on first clock transition

USART_PHASE_2EDGE
USART frame phase on second clock transition

USART Clock Polarity

USART_POLARITY_LOW
Driver enable signal is active high

USART_POLARITY_HIGH
Driver enable signal is active low

USART Error Definition

HAL_USART_ERROR_NONE
No error

HAL_USART_ERROR_PE
Parity error

HAL_USART_ERROR_NE
Noise error

HAL_USART_ERROR_FE
Frame error

HAL_USART_ERROR_ORE

Overrun error

HAL_USART_ERROR_DMA

DMA transfer error

HAL_USART_ERROR_UDR

SPI slave underrun error

HAL_USART_ERROR_INVALID_CALLBACK

Invalid Callback error

HAL_USART_ERROR_RTO

Receiver Timeout error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__: USART handle.

Return value:

- None

__HAL_USART_GET_FLAG

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- __HANDLE__: specifies the USART Handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – USART_FLAG_TXFT TXFIFO threshold flag
  – USART_FLAG_RXFT RXFIFO threshold flag
  – USART_FLAG_RXFF RXFIFO Full flag
  – USART_FLAG_TXFE TXFIFO Empty flag
  – USART_FLAG_REACK Receive enable acknowledge flag
  – USART_FLAG_TEACK Transmit enable acknowledge flag
  – USART_FLAG_BUSY Busy flag
  – USART_FLAG_UDR SPI slave underrun error flag
  – USART_FLAG_TXE Transmit data register empty flag
  – USART_FLAG_TXFNF TXFIFO not full flag
  – USART_FLAG_TC Transmission Complete flag
  – USART_FLAG_RXNE Receive data register not empty flag
  – USART_FLAG_RXFNE RXFIFO not empty flag
  – USART_FLAG_RTOF Receiver Timeout flag
  – USART_FLAG_IDLE Idle Line detection flag
  – USART_FLAG_ORE OverRun Error flag
  – USART_FLAG_NE Noise Error flag
  – USART_FLAG_FE Framing Error flag
  – USART_FLAG_PE Parity Error flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_USART_CLEAR_FLAG

Description:

- Clear the specified USART pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __FLAG__: specifies the flag to check. This parameter can be any combination of the following values:
  – USART_CLEAR_PEF Parity Error Clear Flag
  – USART_CLEAR_FEF Framing Error Clear Flag
  – USART_CLEAR_NEF Noise detected Clear Flag
  – USART_CLEAR_OREF Overrun Error Clear Flag
  – USART_CLEAR_IDLEF IDLE line detected Clear Flag
  – USART_CLEAR_TXFECF TXFIFO empty clear Flag
  – USART_CLEAR_TCF Transmission Complete Clear Flag
  – USART_CLEAR_RTOF Receiver Timeout clear flag
  – USART_CLEAR_UDRF SPI slave underrun error Clear Flag

Return value:

- None

__HAL_USART_CLEAR_PEFLAG

Description:

- Clear the USART PE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_FEFLAG

Description:

- Clear the USART FE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_NEFLAG

Description:

- Clear the USART NE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_OREFLAG

Description:

- Clear the USART ORE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_IDLEFLAG

Description:

- Clear the USART IDLE pending flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_TXFECF

Description:

- Clear the USART TX FIFO empty clear flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_CLEAR_UDRFLAG

Description:

- Clear SPI slave underrun error flag.

Parameters:

- __HANDLE__: specifies the USART Handle.

Return value:

- None

__HAL_USART_ENABLE_IT

Description:

- Enable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to enable. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_PE Parity Error interrupt
  - USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_DISABLE_IT

Description:

- Disable the specified USART interrupt.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to disable. This parameter can be one of the following values:
  – USART_IT_RXFF RXFIFO Full interrupt
  – USART_IT_TXFE TXFIFO Empty interrupt
  – USART_IT_RXFT RXFIFO threshold interrupt
  – USART_IT_TXFT TXFIFO threshold interrupt
  – USART_IT_TXE Transmit Data Register empty interrupt
  – USART_IT_TXFNF TX FIFO not full interrupt
  – USART_IT_TC Transmission complete interrupt
  – USART_IT_RXNE Receive Data register not empty interrupt
  – USART_IT_RXFNE RXFIFO not empty interrupt
  – USART_IT_IDLE Idle line detection interrupt
  – USART_IT_PE Parity Error interrupt
  – USART_IT_ERR Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_USART_GET_IT

Description:

- Check whether the specified USART interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  – USART_IT_RXFF RXFIFO Full interrupt
  – USART_IT_TXFE TXFIFO Empty interrupt
  – USART_IT_RXFT RXFIFO threshold interrupt
  – USART_IT_TXFT TXFIFO threshold interrupt
  – USART_IT_TXE Transmit Data Register empty interrupt
  – USART_IT_TXFNF TX FIFO not full interrupt
  – USART_IT_TC Transmission complete interrupt
  – USART_IT_RXNE Receive Data register not empty interrupt
  – USART_IT_RXFNE RXFIFO not empty interrupt
  – USART_IT_IDLE Idle line detection interrupt
  – USART_IT_ORE OverRun Error interrupt
  – USART_IT_NE Noise Error interrupt
  – USART_IT_FE Framing Error interrupt
  – USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_GET_IT_SOURCE

Description:

- Check whether the specified USART interrupt source is enabled or not.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __INTERRUPT__: specifies the USART interrupt source to check. This parameter can be one of the following values:
  - USART_IT_RXFF RXFIFO Full interrupt
  - USART_IT_TXFE TXFIFO Empty interrupt
  - USART_IT_RXFT RXFIFO threshold interrupt
  - USART_IT_TXFT TXFIFO threshold interrupt
  - USART_IT_TXE Transmit Data Register empty interrupt
  - USART_IT_TXFNF TX FIFO not full interrupt
  - USART_IT_TC Transmission complete interrupt
  - USART_IT_RXNE Receive Data register not empty interrupt
  - USART_IT_RXFNE RXFIFO not empty interrupt
  - USART_IT_IDLE Idle line detection interrupt
  - USART_IT_ORE OverRun Error interrupt
  - USART_IT_NE Noise Error interrupt
  - USART_IT_FE Framing Error interrupt
  - USART_IT_PE Parity Error interrupt

Return value:

- The: new state of __INTERRUPT__ (SET or RESET).

__HAL_USART_CLEAR_IT

Description:

- Clear the specified USART ISR flag, in setting the proper ICR register flag.

Parameters:

- __HANDLE__: specifies the USART Handle.
- __IT_CLEAR__: specifies the interrupt clear register flag that needs to be set to clear the corresponding interrupt. This parameter can be one of the following values:
  - USART_CLEAR_PEF Parity Error Clear Flag
  - USART_CLEAR_FEF Framing Error Clear Flag
  - USART_CLEAR_NEF Noise detected Clear Flag
  - USART_CLEAR_OREF Overrun Error Clear Flag
  - USART_CLEAR_IDLEF IDLE line detected Clear Flag
  - USART_CLEAR_RTOF Receiver timeout clear flag
  - USART_CLEAR_TXFECF TXFIFO empty clear Flag
  - USART_CLEAR_TCF Transmission Complete Clear Flag

Return value:

- None

__HAL_USART_SEND_REQ

Description:
- Set a specific USART request flag.

Parameters:
- __HANDLE__: specifies the USART Handle.
- __REQ__: specifies the request flag to set. This parameter can be one of the following values:
  - USART_RXDATA_FLUSH_REQUEST Receive Data flush Request
  - USART_TXDATA_FLUSH_REQUEST Transmit data flush Request

Return value:
- None

__HAL_USART_ONE_BIT_SAMPLE_ENABLE

Description:
- Enable the USART one bit sample method.

Parameters:
- __HANDLE__: specifies the USART Handle.

Return value:
- None

__HAL_USART_ONE_BIT_SAMPLE_DISABLE

Description:
- Disable the USART one bit sample method.

Parameters:
- __HANDLE__: specifies the USART Handle.

Return value:
- None

__HAL_USART_ENABLE

Description:
- Enable USART.

Parameters:
- __HANDLE__: specifies the USART Handle.

Return value:
- None

__HAL_USART_DISABLE

Description:
- Disable USART.

Parameters:
- __HANDLE__: specifies the USART Handle.

Return value:
- None

USART Flags

USART_FLAG_TXFT

USART TXFIFO threshold flag

USART_FLAG_RXFT

USART RXFIFO threshold flag

USART_FLAG_RXFF
USART RXFIFO Full flag

USART_FLAG_TXFE
USART TXFIFO Empty flag

USART_FLAG_REACK
USART receive enable acknowledge flag

USART_FLAG_TEACK
USART transmit enable acknowledge flag

USART_FLAG_BUSY
USART busy flag

USART_FLAG_UDR
SPI slave underrun error flag

USART_FLAG_TXE
USART transmit data register empty

USART_FLAG_TXFNF
USART TXFIFO not full

USART_FLAG_RTOF
USART receiver timeout flag

USART_FLAG_TC
USART transmission complete

USART_FLAG_RXNE
USART read data register not empty

USART_FLAG_RXFNE
USART RXFIFO not empty

USART_FLAG_IDLE
USART idle flag

USART_FLAG_ORE
USART overrun error

USART_FLAG_NE
USART noise error

USART_FLAG_FE
USART frame error

USART_FLAG_PE
USART parity error

USART Interruption Flags Mask

USART_IT_MASK
USART interruptions flags mask

USART_CR_MASK
USART control register mask

USART_CR_POS

USART control register position

USART_ISR_MASK

USART ISR register mask

USART_ISR_POS

USART ISR register position

USART Interrupts Definition

USART_IT_PE

USART parity error interruption

USART_IT_TXE

USART transmit data register empty interruption

USART_IT_TXFNF

USART TX FIFO not full interruption

USART_IT_TC

USART transmission complete interruption

USART_IT_RXNE

USART read data register not empty interruption

USART_IT_RXFNE

USART RXFIFO not empty interruption

USART_IT_IDLE

USART idle interruption

USART_IT_ERR

USART error interruption

USART_IT_ORE

USART overrun error interruption

USART_IT_NE

USART noise error interruption

USART_IT_FE

USART frame error interruption

USART_IT_RXFF

USART RXFIFO full interruption

USART_IT_TXFE

USART TXFIFO empty interruption

USART_IT_RXFT

USART RXFIFO threshold reached interruption

USART_IT_TXFT

USART TXFIFO threshold reached interruption

USART Interruption Clear Flags

USART_CLEAR_PEF

Parity Error Clear Flag

USART_CLEAR_FEF

Framing Error Clear Flag

USART_CLEAR_NEF

Noise Error detected Clear Flag

USART_CLEAR_OREF

OverRun Error Clear Flag

USART_CLEAR_IDLEF

IDLE line detected Clear Flag

USART_CLEAR_TCF

Transmission Complete Clear Flag

USART_CLEAR_UDRF

SPI slave underrun error Clear Flag

USART_CLEAR_TXFECF

TXFIFO Empty Clear Flag

USART_CLEAR_RTOF

USART receiver timeout clear flag

USART Last Bit

USART_LASTBIT_DISABLE

USART frame last data bit clock pulse not output to SCLK pin

USART_LASTBIT_ENABLE

USART frame last data bit clock pulse output to SCLK pin

USART Mode

USART_MODE_RX

RX mode

USART_MODE_TX

TX mode

USART_MODE_TX_RX

RX and TX mode

USART Parity

USART_PARITY_NONE

No parity

USART_PARITY_EVEN

Even parity

USART_PARITY_ODD

Odd parity

USART Request Parameters

USART_RXDATA_FLUSH_REQUEST

Receive Data flush Request

USART_TXDATA_FLUSH_REQUEST

Transmit data flush Request

USART Number of Stop Bits

USART_STOPBITS_0_5

USART frame with 0.5 stop bit

USART_STOPBITS_1

USART frame with 1 stop bit

USART_STOPBITS_1_5

USART frame with 1.5 stop bits

USART_STOPBITS_2

USART frame with 2 stop bits

87 HAL USART Extension Driver

87.1 USARTEx Firmware driver registers structures

87.1.1 USART_AutonomousModeConfTypeDef

USART_AutonomousModeConfTypeDef is defined in the stm32u5xx_hal_usart_ex.h

Data Fields

- ***uint32_t AutonomousModeState***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerPolarity***
- ***uint32_t DataSize***
- ***uint32_t IdleFrame***

Field Documentation

- ***uint32_t USART_AutonomousModeConfTypeDef::AutonomousModeState***
  Specifies the autonomous mode state.This parameter can be a value of ***USARTEx_Autonomous_mode***.
- ***uint32_t USART_AutonomousModeConfTypeDef::TriggerSelection***
  Specifies which trigger will activate the Transmission automatically. This parameter can be a value of ***USARTEx_Autonomous_Trigger_selection***
- ***uint32_t USART_AutonomousModeConfTypeDef::TriggerPolarity***
  Specifies the autonomous mode trigger signal polarity. This parameter can be a value of ***USARTEx_Autonomous_Trigger_Polarity***
- ***uint32_t USART_AutonomousModeConfTypeDef::DataSize***
  Specifies the transmitted data size in byte
- ***uint32_t USART_AutonomousModeConfTypeDef::IdleFrame***
  Specifies whether the IDLE frame transmission is enabled or disabled. This parameter can be a value of ***USARTEx_Autonomous_IDLE_FRAME***.

87.2 USARTEx Firmware driver API description

The following section lists the various functions of the USARTEx library.

87.2.1 USART peripheral extended features

87.2.2 IO operation functions

This section contains the following APIs:

- ***HAL_USARTEx_RxFifoFullCallback()***
- ***HAL_USARTEx_TxFifoEmptyCallback()***

87.2.3 Peripheral Control functions

This section provides the following functions:

- HAL_USARTEx_EnableSPISlaveMode() API enables the SPI slave mode
- HAL_USARTEx_DisableSPISlaveMode() API disables the SPI slave mode
- HAL_USARTEx_ConfigNSS API configures the Slave Select input pin (NSS)
- HAL_USARTEx_EnableFifoMode() API enables the FIFO mode
- HAL_USARTEx_DisableFifoMode() API disables the FIFO mode
- HAL_USARTEx_SetTxFifoThreshold() API sets the TX FIFO threshold
- HAL_USARTEx_SetRxFifoThreshold() API sets the RX FIFO threshold

This section contains the following APIs:

- ***HAL_USARTEx_EnableSlaveMode()***
- ***HAL_USARTEx_DisableSlaveMode()***

- *HAL_USARTEx_ConfigNSS()*
- *HAL_USARTEx_EnableFifoMode()*
- *HAL_USARTEx_DisableFifoMode()*
- *HAL_USARTEx_SetTxFifoThreshold()*
- *HAL_USARTEx_SetRxFifoThreshold()*
- *HAL_USARTEx_SetConfigAutonomousMode()*
- *HAL_USARTEx_GetConfigAutonomousMode()*
- *HAL_USARTEx_ClearConfigAutonomousMode()*

87.2.4 Detailed description of functions

HAL_USARTEx_RxFifoFullCallback

Function name

void HAL_USARTEx_RxFifoFullCallback (USART_HandleTypeDef * husart)

Function description

USART RX Fifo full callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_TxFifoEmptyCallback

Function name

void HAL_USARTEx_TxFifoEmptyCallback (USART_HandleTypeDef * husart)

Function description

USART TX Fifo empty callback.

Parameters

- **husart:** USART handle.

Return values

- **None:**

HAL_USARTEx_EnableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableSlaveMode (USART_HandleTypeDef * husart)

Function description

Enable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

Notes

- When the USART operates in SPI slave mode, it handles data flow using the serial interface clock derived from the external SCLK signal provided by the external master SPI device.
- In SPI slave mode, the USART must be enabled before starting the master communications (or between frames while the clock is stable). Otherwise, if the USART slave is enabled while the master is in the middle of a frame, it will become desynchronized with the master.
- The data register of the slave needs to be ready before the first edge of the communication clock or before the end of the ongoing communication, otherwise the SPI slave will transmit zeros.

HAL_USARTEx_DisableSlaveMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableSlaveMode (USART_HandleTypeDef * husart)

Function description

Disable the SPI slave mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_ConfigNSS

Function name

HAL_StatusTypeDef HAL_USARTEx_ConfigNSS (USART_HandleTypeDef * husart, uint32_t NSSConfig)

Function description

Configure the Slave Select input pin (NSS).

Parameters

- **husart:** USART handle.
- **NSSConfig:** NSS configuration. This parameter can be one of the following values:
  – USART_NSS_HARD
  – USART_NSS_SOFT

Return values

- **HAL:** status

Notes

- Software NSS management: SPI slave will always be selected and NSS input pin will be ignored.
- Hardware NSS management: the SPI slave selection depends on NSS input pin. The slave is selected when NSS is low and deselected when NSS is high.

HAL_USARTEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_EnableFifoMode (USART_HandleTypeDef * husart)

Function description

Enable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_USARTEx_DisableFifoMode (USART_HandleTypeDef * husart)

Function description

Disable the FIFO mode.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

HAL_USARTEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetTxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  - USART_TXFIFO_THRESHOLD_1_8
  - USART_TXFIFO_THRESHOLD_1_4
  - USART_TXFIFO_THRESHOLD_1_2
  - USART_TXFIFO_THRESHOLD_3_4
  - USART_TXFIFO_THRESHOLD_7_8
  - USART_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_USARTEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_USARTEx_SetRxFifoThreshold (USART_HandleTypeDef * husart, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **husart:** USART handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  – USART_RXFIFO_THRESHOLD_1_8
  – USART_RXFIFO_THRESHOLD_1_4
  – USART_RXFIFO_THRESHOLD_1_2
  – USART_RXFIFO_THRESHOLD_3_4
  – USART_RXFIFO_THRESHOLD_7_8
  – USART_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_USARTEx_SetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_USARTEx_SetConfigAutonomousMode (USART_HandleTypeDef * husart, const USART_AutonomousModeConfTypeDef * sConfig)

Function description

Set autonomous mode Configuration.

Parameters

- **husart:** USART handle.
- **sConfig:** Autonomous mode structure parameters.

Return values

- **HAL:** status

HAL_USARTEx_GetConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_USARTEx_GetConfigAutonomousMode (const USART_HandleTypeDef * husart, USART_AutonomousModeConfTypeDef * sConfig)

Function description

Get autonomous mode Configuration.

Parameters

- **husart:** USART handle.
- **sConfig:** Autonomous mode structure parameters.

Return values

- **HAL:** status

HAL_USARTEx_ClearConfigAutonomousMode

Function name

HAL_StatusTypeDef HAL_USARTEx_ClearConfigAutonomousMode (USART_HandleTypeDef * husart)

Function description

Clear autonomous mode Configuration.

Parameters

- **husart:** USART handle.

Return values

- **HAL:** status

87.3 USARTEx Firmware driver defines

The following section lists the various define and macros of the module.

87.3.1 USARTEx

USARTEx

USARTEx Autonomous IDLE Frame

USART_IDLE_FRAME_ENABLE

IDLE Frame sent after enabling the transmitter

USART_IDLE_FRAME_DISABLE

IDLE Frame not sent after enabling the transmitter

USARTEx Autonomous Mode

USART_AUTONOMOUS_MODE_DISABLE

Autonomous mode disable

USART_AUTONOMOUS_MODE_ENABLE

Autonomous mode enable

USARTEx Autonomous Trigger Polarity

USART_TRIG_POLARITY_RISING

USART triggered on rising edge

USART_TRIG_POLARITY_FALLING

USART triggered on falling edge

USARTEx Autonomous trigger selection

USART_GPDMA1_CH0_TCF_TRG

USART GPDMA1 channel0 Internal Trigger

USART_GPDMA1_CH1_TCF_TRG

USART GPDMA1 channel1 Internal Trigger

USART_GPDMA1_CH2_TCF_TRG

USART GPDMA1 channel2 Internal Trigger

USART_GPDMA1_CH3_TCF_TRG

USART GPDMA1 channel3 Internal Trigger

USART_EXTI_LINE6_TRG

USART EXTI line 6 Internal Trigger

USART_EXTI_LINE9_TRG

USART EXTI line 9 Internal Trigger

USART_LPTIM1_OUT_TRG

USART LPTIM1 out Internal Trigger

USART_LPTIM2_OUT_TRG

USART LPTIM2 out Internal Trigger

USART_COMP1_OUT_TRG

USART COMP1 out Internal Trigger

USART_COMP2_OUT_TRG

USART COMP2 out Internal Trigger

USART_RTC_ALRA_TRG

USART RTC alarm Internal Trigger

USART_RTC_WUT_TRG

USART RTC wakeup Internal Trigger

USARTEx FIFO mode

USART_FIFOMODE_DISABLE

FIFO mode disable

USART_FIFOMODE_ENABLE

FIFO mode enable

USARTEx RXFIFO threshold level

USART_RXFIFO_THRESHOLD_1_8

RXFIFO FIFO reaches 1/8 of its depth

USART_RXFIFO_THRESHOLD_1_4

RXFIFO FIFO reaches 1/4 of its depth

USART_RXFIFO_THRESHOLD_1_2

RXFIFO FIFO reaches 1/2 of its depth

USART_RXFIFO_THRESHOLD_3_4

RXFIFO FIFO reaches 3/4 of its depth

USART_RXFIFO_THRESHOLD_7_8

RXFIFO FIFO reaches 7/8 of its depth

USART_RXFIFO_THRESHOLD_8_8

RXFIFO FIFO becomes full

USARTEx Synchronous Slave mode enable

USART_SLAVEMODE_DISABLE

USART SPI Slave Mode Enable

USART_SLAVEMODE_ENABLE

USART SPI Slave Mode Disable

USARTEx Slave Select Management

USART_NSS_HARD

SPI slave selection depends on NSS input pin

USART_NSS_SOFT

SPI slave is always selected and NSS input pin is ignored

USARTEx TXFIFO threshold level

USART_TXFIFO_THRESHOLD_1_8

TXFIFO reaches 1/8 of its depth

USART_TXFIFO_THRESHOLD_1_4

TXFIFO reaches 1/4 of its depth

USART_TXFIFO_THRESHOLD_1_2

TXFIFO reaches 1/2 of its depth

USART_TXFIFO_THRESHOLD_3_4

TXFIFO reaches 3/4 of its depth

USART_TXFIFO_THRESHOLD_7_8

TXFIFO reaches 7/8 of its depth

USART_TXFIFO_THRESHOLD_8_8

TXFIFO becomes empty

USARTEx Word Length

USART_WORDLENGTH_7B

7-bit long USART frame

USART_WORDLENGTH_8B

8-bit long USART frame

USART_WORDLENGTH_9B

9-bit long USART frame

88 HAL WWDG Generic Driver

88.1 WWDG Firmware driver registers structures

88.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32u5xx_hal_wwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Window***
- ***uint32_t Counter***
- ***uint32_t EWIMode***

Field Documentation

- ***uint32_t WWDG_InitTypeDef::Prescaler***
  Specifies the prescaler value of the WWDG. This parameter can be a value of ***WWDG_Prescaler***
- ***uint32_t WWDG_InitTypeDef::Window***
  Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::Counter***
  Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F
- ***uint32_t WWDG_InitTypeDef::EWIMode***
  Specifies if WWDG Early Wakeup Interrupt is enable or not. This parameter can be a value of ***WWDG_EWI_Mode***

88.1.2 __WWDG_HandleTypeDef

__WWDG_HandleTypeDef is defined in the stm32u5xx_hal_wwdg.h

Data Fields

- ***WWDG_TypeDef * Instance***
- ***WWDG_InitTypeDef Init***
- ***void(* EwiCallback***
- ***void(* MspInitCallback***

Field Documentation

- ***WWDG_TypeDef* __WWDG_HandleTypeDef::Instance***
  Register base address
- ***WWDG_InitTypeDef __WWDG_HandleTypeDef::Init***
  WWDG required parameters
- ***void(* __WWDG_HandleTypeDef::EwiCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Early WakeUp Interrupt callback
- ***void(* __WWDG_HandleTypeDef::MspInitCallback)(struct __WWDG_HandleTypeDef *hwwdg)***
  WWDG Msp Init callback

88.2 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

88.2.1 WWDG Specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (T[6;0] downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls down from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.

- Once enabled the WWDG cannot be disabled except by a system reset.
- If required by application, an Early Wakeup Interrupt can be triggered in order to be warned before WWDG expiration. The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. When the downcounter reaches 0x40, interrupt occurs. This mechanism requires WWDG interrupt line to be enabled in NVIC. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- WWDGRST flag in RCC CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- WWDG clock (Hz) = PCLK1 / (4096 * Prescaler)
- WWDG timeout (mS) = 1000 * (T[5;0] + 1) / WWDG clock (Hz) where T[5;0] are the lowest 6 bits of Counter.
- WWDG Counter refresh is allowed between the following limits :
  – min time (mS) = 1000 * (Counter - Window) / WWDG clock
  – max time (mS) = 1000 * (Counter - 0x40) / WWDG clock
- Typical values:
  – Counter min (T[5;0] = 0x00) at 56MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 73.14us
  – Counter max (T[5;0] = 0x3F) at 56MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 599.18ms

88.2.2 How to use this driver

Common driver usage

- Enable WWDG APB1 clock using __HAL_RCC_WWDG_CLK_ENABLE().
- Configure the WWDG prescaler, refresh window value, counter value and early interrupt status using HAL_WWDG_Init() function. This will automatically enable WWDG and start its downcounter. Time reference can be taken from function exit. Care must be taken to provide a counter value greater than 0x40 to prevent generation of immediate reset.
- If the Early Wakeup Interrupt (EWI) feature is enabled, an interrupt is generated when the counter reaches 0x40. When HAL_WWDG_IRQHandler is triggered by the interrupt service routine, flag will be automatically cleared and HAL_WWDG_WakeupCallback user callback will be executed. User can add his own code by customization of callback HAL_WWDG_WakeupCallback.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using HAL_WWDG_Refresh() function. This operation must occur only when the counter is lower than the refresh window value already programmed.

Callback registration

The compilation define USE_HAL_WWDG_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Functions HAL_WWDG_RegisterCallback() to register a user callback.

- Function HAL_WWDG_RegisterCallback() allows to register following callbacks:
  – EwiCallback : callback for Early WakeUp Interrupt.
  – MspInitCallback : WWDG MspInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.
- Use function HAL_WWDG_UnRegisterCallback() to reset a callback to the default weak (surcharged) function. HAL_WWDG_UnRegisterCallback() takes as parameters the HAL peripheral handle and the Callback ID. This function allows to reset following callbacks:
  – EwiCallback : callback for Early WakeUp Interrupt.
  – MspInitCallback : WWDG MspInit.

When calling HAL_WWDG_Init function, callbacks are reset to the corresponding legacy weak (surcharged) functions: HAL_WWDG_EarlyWakeupCallback() and HAL_WWDG_MspInit() only if they have not been registered before.

When compilation define USE_HAL_WWDG_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

WWDG HAL driver macros list

Below the list of available macros in WWDG HAL driver.
- __HAL_WWDG_ENABLE: Enable the WWDG peripheral
- __HAL_WWDG_GET_FLAG: Get the selected WWDG's flag status
- __HAL_WWDG_CLEAR_FLAG: Clear the WWDG's pending flags
- __HAL_WWDG_ENABLE_IT: Enable the WWDG early wakeup interrupt

88.2.3 Initialization and Configuration functions

This section provides functions allowing to:
- Initialize and start the WWDG according to the specified parameters in the WWDG_InitTypeDef of associated handle.
- Initialize the WWDG MSP.

This section contains the following APIs:
- *HAL_WWDG_Init()*
- *HAL_WWDG_MspInit()*
- *HAL_WWDG_RegisterCallback()*
- *HAL_WWDG_UnRegisterCallback()*

88.2.4 IO operation functions

This section provides functions allowing to:
- Refresh the WWDG.
- Handle WWDG interrupt request and associated function callback.

This section contains the following APIs:
- *HAL_WWDG_Refresh()*
- *HAL_WWDG_IRQHandler()*
- *HAL_WWDG_EarlyWakeupCallback()*

88.2.5 Detailed description of functions

HAL_WWDG_Init

Function name

HAL_StatusTypeDef HAL_WWDG_Init (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG according to the specified.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_MspInit

Function name

void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)

Function description

Initialize the WWDG MSP.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- When rewriting this function in user file, mechanism may be added to avoid multiple initialize when HAL_WWDG_Init function is called again to change parameters.

HAL_WWDG_RegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_RegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID, pWWDG_CallbackTypeDef pCallback)

Function description

Register a User WWDG Callback To be used instead of the weak (surcharged) predefined callback.

Parameters

- **hwwdg:** WWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_WWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  – HAL_WWDG_MSPINIT_CB_ID MspInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **status:**

HAL_WWDG_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_WWDG_UnRegisterCallback (WWDG_HandleTypeDef * hwwdg, HAL_WWDG_CallbackIDTypeDef CallbackID)

Function description

Unregister a WWDG Callback WWDG Callback is redirected to the weak (surcharged) predefined callback.

Parameters

- **hwwdg:** WWDG handle
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  – HAL_WWDG_EWI_CB_ID Early WakeUp Interrupt Callback ID
  – HAL_WWDG_MSPINIT_CB_ID MspInit callback ID

Return values

- **status:**

HAL_WWDG_Refresh

Function name

HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg)

Function description

Refresh the WWDG.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **HAL:** status

HAL_WWDG_IRQHandler

Function name

void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)

Function description

Handle WWDG interrupt request.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

Notes

- The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled by calling HAL_WWDG_Init function with EWIMode set to WWDG_EWI_ENABLE. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

HAL_WWDG_EarlyWakeupCallback

Function name

void HAL_WWDG_EarlyWakeupCallback (WWDG_HandleTypeDef * hwwdg)

Function description

WWDG Early Wakeup callback.

Parameters

- **hwwdg:** pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- **None:**

88.3 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

88.3.1 WWDG

WWDG

WWDG Early Wakeup Interrupt Mode

WWDG_EWI_DISABLE

EWI Disable

WWDG_EWI_ENABLE

EWI Enable

WWDG Exported Macros

__HAL_WWDG_ENABLE

Description:

- Enable the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE_IT

Description:

- Enable the WWDG early wakeup interrupt.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt to enable. This parameter can be one of the following values:
  - WWDG_IT_EWI: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

__HAL_WWDG_GET_IT

Description:

- Check whether the selected WWDG interrupt has occurred or not.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the it to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt IT

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_IT

Description:

- Clear the WWDG interrupt pending bits.

Parameters:

- __HANDLE__: WWDG handle
- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

__HAL_WWDG_GET_FLAG

Description:

- Check whether the specified WWDG flag is set or not.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- The: new state of WWDG_FLAG (SET or RESET).

__HAL_WWDG_CLEAR_FLAG

Description:

- Clear the WWDG's pending flags.

Parameters:

- __HANDLE__: WWDG handle
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  – WWDG_FLAG_EWIF: Early wakeup interrupt flag

Return value:

- None

__HAL_WWDG_GET_IT_SOURCE

Description:

- Check whether the specified WWDG interrupt source is enabled or not.

Parameters:

- __HANDLE__: WWDG Handle.
- __INTERRUPT__: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
  – WWDG_IT_EWI: Early Wakeup Interrupt

Return value:

- state: of __INTERRUPT__ (TRUE or FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF

Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI

Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

89 HAL XSPI Generic Driver

89.1 XSPI Firmware driver registers structures

89.1.1 XSPI_InitTypeDef

XSPI_InitTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t FifoThresholdByte***
- ***uint32_t MemoryMode***
- ***uint32_t MemoryType***
- ***uint32_t MemorySize***
- ***uint32_t ChipSelectHighTimeCycle***
- ***uint32_t FreeRunningClock***
- ***uint32_t ClockMode***
- ***uint32_t WrapSize***
- ***uint32_t ClockPrescaler***
- ***uint32_t SampleShifting***
- ***uint32_t DelayHoldQuarterCycle***
- ***uint32_t ChipSelectBoundary***
- ***uint32_t DelayBlockBypass***
- ***uint32_t MaxTran***
- ***uint32_t Refresh***

Field Documentation

- ***uint32_t XSPI_InitTypeDef::FifoThresholdByte***
  This is the threshold used by the Peripheral to generate the interrupt indicating that data are available in reception or free place is available in transmission. For OCTOSPI, this parameter can be a value between 1 and 32. For HSPI, this parameter can be a value between 1 and 64
- ***uint32_t XSPI_InitTypeDef::MemoryMode***
  It Specifies the memory mode. This parameter can be a value of ***XSPI_MemoryMode***
- ***uint32_t XSPI_InitTypeDef::MemoryType***
  It indicates the external device type connected to the XSPI. This parameter can be a value of ***XSPI_MemoryType***
- ***uint32_t XSPI_InitTypeDef::MemorySize***
  It defines the size of the external device connected to the XSPI, it corresponds to the number of address bits required to access the external device. This parameter can be a value of ***XSPI_MemorySize***
- ***uint32_t XSPI_InitTypeDef::ChipSelectHighTimeCycle***
  It defines the minimum number of clocks which the chip select must remain high between commands. This parameter can be a value between 1 and 64U
- ***uint32_t XSPI_InitTypeDef::FreeRunningClock***
  It enables or not the free running clock. This parameter can be a value of ***XSPI_FreeRunningClock***
- ***uint32_t XSPI_InitTypeDef::ClockMode***
  It indicates the level of clock when the chip select is released. This parameter can be a value of ***XSPI_ClockMode***
- ***uint32_t XSPI_InitTypeDef::WrapSize***
  It indicates the wrap-size corresponding the external device configuration. This parameter can be a value of ***XSPI_WrapSize***
- ***uint32_t XSPI_InitTypeDef::ClockPrescaler***
  It specifies the prescaler factor used for generating the external clock based on the AHB clock. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_InitTypeDef::SampleShifting***
  It allows to delay to 1/2 cycle the data sampling in order to take in account external signal delays. This parameter can be a value of ***XSPI_SampleShifting***

- ***uint32_t XSPI_InitTypeDef::DelayHoldQuarterCycle***
  It allows to hold to 1/4 cycle the data. This parameter can be a value of ***XSPI_DelayHoldQuarterCycle***
- ***uint32_t XSPI_InitTypeDef::ChipSelectBoundary***
  It enables the transaction boundary feature and defines the boundary of bytes to release the chip select. This parameter can be a value of ***XSPI_ChipSelectBoundary***
- ***uint32_t XSPI_InitTypeDef::DelayBlockBypass***
  It enables the delay block bypass, so the sampling is not affected by the delay block. This parameter can be a value of ***XSPI_DelayBlockBypass***
- ***uint32_t XSPI_InitTypeDef::MaxTran***
  It enables the communication regulation feature. The chip select is released every MaxTran+1 bytes when the other XSPI request the access to the bus. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_InitTypeDef::Refresh***
  It enables the refresh rate feature. The chip select is released every Refresh+1 clock cycles. This parameter can be a value between 0 and 0xFFFFFFFF

89.1.2 __XSPI_HandleTypeDef

__XSPI_HandleTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***XSPI_TypeDef * Instance***
- ***XSPI_InitTypeDef Init***
- ***uint8_t * pBuffPtr***
- ***__IO uint32_t XferSize***
- ***__IO uint32_t XferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t Timeout***
- ***void(* ErrorCallback***
- ***void(* AbortCpltCallback***
- ***void(* FifoThresholdCallback***
- ***void(* CmdCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* StatusMatchCallback***
- ***void(* TimeOutCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***XSPI_TypeDef* __XSPI_HandleTypeDef::Instance***
  XSPI registers base address
- ***XSPI_InitTypeDef __XSPI_HandleTypeDef::Init***
  XSPI initialization parameters
- ***uint8_t* __XSPI_HandleTypeDef::pBuffPtr***
  Address of the XSPI buffer for transfer
- ***__IO uint32_t __XSPI_HandleTypeDef::XferSize***
  Number of data to transfer
- ***__IO uint32_t __XSPI_HandleTypeDef::XferCount***
  Counter of data transferred
- ***DMA_HandleTypeDef* __XSPI_HandleTypeDef::hdmatx***
  Handle of the DMA channel used for transmit

- ***DMA_HandleTypeDef* __XSPI_HandleTypeDef::hdmarx***
  Handle of the DMA channel used for receive
- ***__IO uint32_t __XSPI_HandleTypeDef::State***
  Internal state of the XSPI HAL driver
- ***__IO uint32_t __XSPI_HandleTypeDef::ErrorCode***
  Error code in case of HAL driver internal error
- ***uint32_t __XSPI_HandleTypeDef::Timeout***
  Timeout used for the XSPI external device access
- ***void(* __XSPI_HandleTypeDef::ErrorCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::AbortCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::FifoThresholdCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::CmdCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::RxCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TxCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::RxHalfCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TxHalfCpltCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::StatusMatchCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::TimeOutCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::MspInitCallback)(struct __XSPI_HandleTypeDef *hxspi)***
- ***void(* __XSPI_HandleTypeDef::MspDeInitCallback)(struct __XSPI_HandleTypeDef *hxspi)***

89.1.3 XSPI_RegularCmdTypeDef

XSPI_RegularCmdTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t OperationType***
- ***uint32_t IOSelect***
- ***uint32_t Instruction***
- ***uint32_t InstructionMode***
- ***uint32_t InstructionWidth***
- ***uint32_t InstructionDTRMode***
- ***uint32_t Address***
- ***uint32_t AddressMode***
- ***uint32_t AddressWidth***
- ***uint32_t AddressDTRMode***
- ***uint32_t AlternateBytes***
- ***uint32_t AlternateBytesMode***
- ***uint32_t AlternateBytesWidth***
- ***uint32_t AlternateBytesDTRMode***
- ***uint32_t DataMode***
- ***uint32_t DataLength***
- ***uint32_t DataDTRMode***
- ***uint32_t DummyCycles***
- ***uint32_t DQSMode***
- ***uint32_t SIOOMode***

Field Documentation

- ***uint32_t XSPI_RegularCmdTypeDef::OperationType***
  It indicates if the configuration applies to the common registers or to the registers for the write operation (these registers are only used for memory-mapped mode). This parameter can be a value of ***XSPI_OperationType***
- ***uint32_t XSPI_RegularCmdTypeDef::IOSelect***
  It indicates the IOs used to exchange data with external memory. This parameter can be a value of ***XSPI_IOSelect***

- ***uint32_t XSPI_RegularCmdTypeDef::Instruction***
  It contains the instruction to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionMode***
  It indicates the mode of the instruction. This parameter can be a value of ***XSPI_InstructionMode***
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionWidth***
  It indicates the width of the instruction. This parameter can be a value of ***XSPI_InstructionWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::InstructionDTRMode***
  It enables or not the DTR mode for the instruction phase. This parameter can be a value of ***XSPI_InstructionDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_RegularCmdTypeDef::AddressMode***
  It indicates the address mode. Address mode precises number of lines for address (except no address). This parameter can be a value of ***XSPI_AddressMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AddressWidth***
  It indicates the width of the address. This parameter can be a value of ***XSPI_AddressWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::AddressDTRMode***
  It enables or not the DTR mode for the address phase. This parameter can be a value of ***XSPI_AddressDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytes***
  It contains the alternate bytes to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesMode***
  It indicates the mode of the alternate bytes. This parameter can be a value of ***XSPI_AlternateBytesMode***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesWidth***
  It indicates the width of the alternate bytes. This parameter can be a value of ***XSPI_AlternateBytesWidth***
- ***uint32_t XSPI_RegularCmdTypeDef::AlternateBytesDTRMode***
  It enables or not the DTR mode for the alternate bytes phase. This parameter can be a value of ***XSPI_AlternateBytesDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DataMode***
  It indicates the data mode. Data mode precises number of lines for data exchange (except no data). This parameter can be a value of ***XSPI_DataMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DataLength***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFFU
- ***uint32_t XSPI_RegularCmdTypeDef::DataDTRMode***
  It enables or not the DTR mode for the data phase. This parameter can be a value of ***XSPI_DataDTRMode***
- ***uint32_t XSPI_RegularCmdTypeDef::DummyCycles***
  It indicates the number of dummy cycles inserted before data phase. This parameter can be a value between 0 and 31U
- ***uint32_t XSPI_RegularCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***XSPI_DQSMode***
- ***uint32_t XSPI_RegularCmdTypeDef::SIOOMode***
  It enables or not the SIOO mode. When SIOO mode enabled, instruction will be sent only once. This parameter can be a value of ***XSPI_SIOOMode***

89.1.4 XSPI_HyperbusCfgTypeDef

XSPI_HyperbusCfgTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t RWRecoveryTimeCycle***
- ***uint32_t AccessTimeCycle***
- ***uint32_t WriteZeroLatency***
- ***uint32_t LatencyMode***

Field Documentation

- ***uint32_t XSPI_HyperbusCfgTypeDef::RWRecoveryTimeCycle***
  It indicates the number of cycles for the device read write recovery time. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_HyperbusCfgTypeDef::AccessTimeCycle***
  It indicates the number of cycles for the device access time. This parameter can be a value between 0 and 255U
- ***uint32_t XSPI_HyperbusCfgTypeDef::WriteZeroLatency***
  It enables or not the latency for the write access. This parameter can be a value of ***XSPI_WriteZeroLatency***
- ***uint32_t XSPI_HyperbusCfgTypeDef::LatencyMode***
  It configures the latency mode. This parameter can be a value of ***XSPI_LatencyMode***

89.1.5 XSPI_HyperbusCmdTypeDef

XSPI_HyperbusCmdTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t AddressSpace***
- ***uint32_t Address***
- ***uint32_t AddressWidth***
- ***uint32_t DataLength***
- ***uint32_t DQSMode***
- ***uint32_t DataMode***

Field Documentation

- ***uint32_t XSPI_HyperbusCmdTypeDef::AddressSpace***
  It indicates the address space accessed by the command. This parameter can be a value of ***XSPI_AddressSpace***
- ***uint32_t XSPI_HyperbusCmdTypeDef::Address***
  It contains the address to be sent to the device. This parameter can be a value between 0 and 0xFFFFFFFF
- ***uint32_t XSPI_HyperbusCmdTypeDef::AddressWidth***
  It indicates the width of the address. This parameter can be a value of ***XSPI_AddressWidth***
- ***uint32_t XSPI_HyperbusCmdTypeDef::DataLength***
  It indicates the number of data transferred with this command. This field is only used for indirect mode. This parameter can be a value between 1 and 0xFFFFFFFF In case of autopolling mode, this parameter can be any value between 1 and 4
- ***uint32_t XSPI_HyperbusCmdTypeDef::DQSMode***
  It enables or not the data strobe management. This parameter can be a value of ***XSPI_DQSMode***
- ***uint32_t XSPI_HyperbusCmdTypeDef::DataMode***
  It indicates the data mode. Data mode precises number of lines for data exchange (except no data). This parameter can be a value of ***XSPI_DataMode***

89.1.6 XSPI_AutoPollingTypeDef

XSPI_AutoPollingTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t MatchValue***
- ***uint32_t MatchMask***
- ***uint32_t MatchMode***
- ***uint32_t AutomaticStop***
- ***uint32_t IntervalTime***

Field Documentation

- ***uint32_t XSPI_AutoPollingTypeDef::MatchValue***
  Specifies the value to be compared with the masked status register to get a match. This parameter can be any value between 0 and 0xFFFFFFFFU

- ***uint32_t XSPI_AutoPollingTypeDef::MatchMask***
  Specifies the mask to be applied to the status bytes received. This parameter can be any value between 0 and 0xFFFFFFFFU
- ***uint32_t XSPI_AutoPollingTypeDef::MatchMode***
  Specifies the method used for determining a match. This parameter can be a value of ***XSPI_MatchMode***
- ***uint32_t XSPI_AutoPollingTypeDef::AutomaticStop***
  Specifies if automatic polling is stopped after a match. This parameter can be a value of ***XSPI_AutomaticStop***
- ***uint32_t XSPI_AutoPollingTypeDef::IntervalTime***
  Specifies the number of clock cycles between two read during automatic polling phases. This parameter can be any value between 0 and 0xFFFFU

89.1.7 XSPI_MemoryMappedTypeDef

XSPI_MemoryMappedTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t TimeOutActivation***
- ***uint32_t TimeoutPeriodClock***

Field Documentation

- ***uint32_t XSPI_MemoryMappedTypeDef::TimeOutActivation***
  Specifies if the timeout counter is enabled to release the chip select. This parameter can be a value of ***XSPI_TimeOutActivation***
- ***uint32_t XSPI_MemoryMappedTypeDef::TimeoutPeriodClock***
  Specifies the number of clock to wait when the FIFO is full before to release the chip select. This parameter can be any value between 0 and 0xFFFFU

89.1.8 XSPIM_CfgTypeDef

XSPIM_CfgTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t ClkPort***
- ***uint32_t DQSPort***
- ***uint32_t NCSPort***
- ***uint32_t IOLowPort***
- ***uint32_t IOHighPort***
- ***uint32_t Req2AckTime***

Field Documentation

- ***uint32_t XSPIM_CfgTypeDef::ClkPort***
  It indicates which port of the XSPI IO Manager is used for the CLK pins. This parameter can be a value between 1 and 8
- ***uint32_t XSPIM_CfgTypeDef::DQSPort***
  It indicates which port of the XSPI IO Manager is used for the DQS pin. This parameter can be a value between 1 and 8, 0 means that signal not used
- ***uint32_t XSPIM_CfgTypeDef::NCSPort***
  It indicates which port of the XSPI IO Manager is used for the NCS pin. This parameter can be a value between 1 and 8
- ***uint32_t XSPIM_CfgTypeDef::IOLowPort***
  It indicates which port of the XSPI IO Manager is used for the IO[3:0] pins. This parameter can be a value of XSPIM_IOPort
- ***uint32_t XSPIM_CfgTypeDef::IOHighPort***
  It indicates which port of the XSPI IO Manager is used for the IO[7:4] pins. This parameter can be a value of XSPIM_IOPort
- ***uint32_t XSPIM_CfgTypeDef::Req2AckTime***
  It indicates the minimum switching duration (in number of clock cycles) expected if some signals are multiplexed in the XSPI IO Manager with the other XSPI. This parameter can be a value between 1 and 256

89.1.9 XSPI_HSCalTypeDef

XSPI_HSCalTypeDef is defined in the stm32u5xx_hal_xspi.h

Data Fields

- ***uint32_t DelayValueType***
- ***uint32_t FineCalibrationUnit***
- ***uint32_t CoarseCalibrationUnit***
- ***uint32_t MaxCalibration***

Field Documentation

- ***uint32_t XSPI_HSCalTypeDef::DelayValueType***
  It indicates which calibration is concerned by the configuration. This parameter can be a value of ***XSPI_DelayType***
- ***uint32_t XSPI_HSCalTypeDef::FineCalibrationUnit***
  It indicates the fine calibration value of the delay. This parameter can be a value between 0 and 0x7FU
- ***uint32_t XSPI_HSCalTypeDef::CoarseCalibrationUnit***
  It indicates the coarse calibration value of the delay. This parameter can be a value between 0 and 0x1FU
- ***uint32_t XSPI_HSCalTypeDef::MaxCalibration***
  It indicates that the calibration is outside the range of DLL master. It applies only when the DelayValueType is HAL_XSPI_CAL_FULL_CYCLE_DELAY. This parameter can be a value of ***XSPI_MaxCal***

89.2 XSPI Firmware driver API description

The following section lists the various functions of the XSPI library.

89.2.1 How to use this driver

Initialization

As prerequisite, fill in the HAL_XSPI_MspInit() :

- Enable OctoSPI/HSPI clocks interface with __HAL_RCC_XSPI_CLK_ENABLE().
- Reset OctoSPI/HSPI Peripheral with __HAL_RCC_XSPI_FORCE_RESET() and __HAL_RCC_XSPI_RELEASE_RESET().
- Enable the clocks for the OctoSPI/HSPI GPIOS with __HAL_RCC_GPIOx_CLK_ENABLE().
- Configure these OctoSPI/HSPI pins in alternate mode using HAL_GPIO_Init().
- If interrupt or DMA mode is used, enable and configure OctoSPI/HSPI global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().
- If DMA mode is used, enable the clocks for the OctoSPI/HSPI DMA channel with __HAL_RCC_DMAx_CLK_ENABLE(), configure DMA with HAL_DMA_Init(), link it with OctoSPI/HSPI handle using __HAL_LINKDMA(), enable and configure DMA channel global interrupt with HAL_NVIC_SetPriority() and HAL_NVIC_EnableIRQ().

Configure the fifo threshold, the memory mode, the memory type, the device size, the CS high time, the free running clock, the clock mode, the wrap size, the clock prescaler, the sample shifting, the hold delay and the CS boundary using the HAL_XSPI_Init() function.

When using Hyperbus, configure the RW recovery time, the access time, the write latency and the latency mode using the HAL_XSPI_HyperbusCfg() function.

Indirect functional mode

In regular mode, configure the command sequence using the HAL_XSPI_Command() or HAL_XSPI_Command_IT() functions :

- Instruction phase : the mode used and if present the size, the instruction opcode and the DTR mode.
- Address phase : the mode used and if present the size, the address value and the DTR mode.
- Alternate-bytes phase : the mode used and if present the size, the alternate bytes values and the DTR mode.
- Dummy-cycles phase : the number of dummy cycles (mode used is same as data phase).
- Data phase : the mode used and if present the number of bytes and the DTR mode.

- Data strobe (DQS) mode : the activation (or not) of this mode
- Sending Instruction Only Once (SIOO) mode : the activation (or not) of this mode.
- IO selection : to access external memory.
- Operation type : always common configuration.

In Hyperbus mode, configure the command sequence using the HAL_XSPI_HyperbusCmd() function :

- Address space : indicate if the access will be done in register or memory
- Address size
- Number of data
- Data strobe (DQS) mode : the activation (or not) of this mode

If no data is required for the command (only for regular mode, not for Hyperbus mode), it is sent directly to the memory :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_CmdCpltCallback() will be called when the transfer is complete.

For the indirect write mode, use HAL_XSPI_Transmit(), HAL_XSPI_Transmit_DMA() or HAL_XSPI_Transmit_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_XSPI_TxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_XSPI_TxHalfCpltCallback() will be called at the half transfer and HAL_XSPI_TxCpltCallback() will be called when the transfer is complete.

For the indirect read mode, use HAL_XSPI_Receive(), HAL_XSPI_Receive_DMA() or HAL_XSPI_Receive_IT() after the command configuration :

- In polling mode, the output of the function is done when the transfer is complete.
- In interrupt mode, HAL_XSPI_FifoThresholdCallback() will be called when the fifo threshold is reached and HAL_XSPI_RxCpltCallback() will be called when the transfer is complete.
- In DMA mode, HAL_XSPI_RxHalfCpltCallback() will be called at the half transfer and HAL_XSPI_RxCpltCallback() will be called when the transfer is complete.

Auto-polling functional mode

Configure the command sequence by the same way than the indirect mode

Configure the auto-polling functional mode using the HAL_XSPI_AutoPolling() or HAL_XSPI_AutoPolling_IT() functions :

- The size of the status bytes, the match value, the mask used, the match mode (OR/AND), the polling interval and the automatic stop activation.

After the configuration :

- In polling mode, the output of the function is done when the status match is reached. The automatic stop is activated to avoid an infinite loop.
- In interrupt mode, HAL_XSPI_StatusMatchCallback() will be called each time the status match is reached.

Memory-mapped functional mode

Configure the command sequence by the same way than the indirect mode except for the operation type in regular mode :

- Operation type equals to read configuration : the command configuration applies to read access in memory-mapped mode
- Operation type equals to write configuration : the command configuration applies to write access in memory-mapped mode
- Both read and write configuration should be performed before activating memory-mapped mode

Configure the memory-mapped functional mode using the HAL_XSPI_MemoryMapped() functions :

- The timeout activation and the timeout period.

After the configuration, the OctoSPI/HSPI will be used as soon as an access on the AHB is done on the address range. HAL_XSPI_TimeOutCallback() will be called when the timeout expires.

Errors management and abort functionality

HAL_XSPI_GetError() function gives the error raised during the last operation.

HAL_XSPI_Abort() and HAL_XSPI_AbortIT() functions aborts any on-going operation and flushes the fifo :

- In polling mode, the output of the function is done when the transfer complete bit is set and the busy bit cleared.
- In interrupt mode, HAL_XSPI_AbortCpltCallback() will be called when the transfer complete bit is set.

Control functions

HAL_XSPI_GetState() function gives the current state of the HAL XSPI driver.

HAL_XSPI_SetTimeout() function configures the timeout value used in the driver.

HAL_XSPI_SetFifoThreshold() function configures the threshold on the Fifo of the OctoSPI/HSPI Peripheral.

HAL_XSPI_SetMemoryType() function configures the type of the external memory.

HAL_XSPI_SetDeviceSize() function configures the size of the external memory.

HAL_XSPI_SetClockPrescaler() function configures the clock prescaler of the OctoSPI/HSPI Peripheral.

HAL_XSPI_GetFifoThreshold() function gives the current of the Fifo's threshold

IO manager configuration functions (if available)

HAL_XSPIM_Config() function configures the IO manager (if available) for the XSPI instance.

Delay Block functions

The delay block (DLYB) is used to generate an output clock that is dephased from the input clock.

- The delay line length can be Configure to one period of the Input clock with HAL_XSPI_DLYB_GetClockPeriod().
- The phase of the output clock can be programmed directly with HAL_XSPI_DLYB_SetConfig().
- The phase of the output clock can be got with HAL_XSPI_DLYB_GetConfig().

High-speed interface and calibration functions

The purpose of High-speed interface is primary to shift data or data strobe by one quarter of octal bus clock period, with a correct timing accuracy. DLL must be calibrated versus this clock period. The calibration process is automatically enabled when one of the three conditions below is met:

- The HSPI exits Reset state.
- The Prescaler is set.
- The configuration of communication is set.

HAL_XSPI_GetDelayValue() function Get the delay values of the high-speed interface DLLs..

HAL_XSPI_SetDelayValue() function Set the delay values of the high-speed interface DLLs..

Callback registration

The compilation define USE_HAL_XSPI_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks.

Use function HAL_XSPI_RegisterCallback() to register a user callback, it allows to register following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.

- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : XSPI MspInit.
- MspDeInitCallback : XSPI MspDeInit.

This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

Use function HAL_XSPI_UnRegisterCallback() to reset a callback to the default weak (overridden) function. It allows to reset following callbacks:

- ErrorCallback : callback when error occurs.
- AbortCpltCallback : callback when abort is completed.
- FifoThresholdCallback : callback when the fifo threshold is reached.
- CmdCpltCallback : callback when a command without data is completed.
- RxCpltCallback : callback when a reception transfer is completed.
- TxCpltCallback : callback when a transmission transfer is completed.
- RxHalfCpltCallback : callback when half of the reception transfer is completed.
- TxHalfCpltCallback : callback when half of the transmission transfer is completed.
- StatusMatchCallback : callback when a status match occurs.
- TimeOutCallback : callback when the timeout perioed expires.
- MspInitCallback : XSPI MspInit.
- MspDeInitCallback : XSPI MspDeInit.

This function) takes as parameters the HAL peripheral handle and the Callback ID.

By default, after the HAL_XSPI_Init() and if the state is HAL_XSPI_STATE_RESET all callbacks are reset to the corresponding legacy weak (overridden) functions. Exception done for MspInit and MspDeInit callbacks that are respectively reset to the legacy weak (overridden) functions in the HAL_XSPI_Init() and HAL_XSPI_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, the HAL_XSPI_Init() and HAL_XSPI_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in READY state only. Exception done for MspInit/MspDeInit callbacks that can be registered/unregistered in READY or RESET state, thus registered (user) MspInit/DeInit callbacks can be used during the Init/DeInit. In that case first register the MspInit/MspDeInit user callbacks using HAL_XSPI_RegisterCallback() before calling HAL_XSPI_DeInit() or HAL_XSPI_Init() function.

When The compilation define USE_HAL_XSPI_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (overridden) callbacks are used.

89.2.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to :

- Initialize the XSPI.
- De-initialize the XSPI.

This section contains the following APIs:

- *HAL_XSPI_Init()*
- *HAL_XSPI_MspInit()*
- *HAL_XSPI_DeInit()*
- *HAL_XSPI_MspDeInit()*

89.2.3 IO operation functions

This subsection provides a set of functions allowing to :

- Handle the interrupts.
- Handle the command sequence (regular and Hyperbus).
- Handle the Hyperbus configuration.
- Transmit data in blocking, interrupt or DMA mode.
- Receive data in blocking, interrupt or DMA mode.
- Manage the auto-polling functional mode.
- Manage the memory-mapped functional mode.

This section contains the following APIs:
- *HAL_XSPI_IRQHandler()*
- *HAL_XSPI_Command()*
- *HAL_XSPI_Command_IT()*
- *HAL_XSPI_HyperbusCfg()*
- *HAL_XSPI_HyperbusCmd()*
- *HAL_XSPI_Transmit()*
- *HAL_XSPI_Receive()*
- *HAL_XSPI_Transmit_IT()*
- *HAL_XSPI_Receive_IT()*
- *HAL_XSPI_Transmit_DMA()*
- *HAL_XSPI_Receive_DMA()*
- *HAL_XSPI_AutoPolling()*
- *HAL_XSPI_AutoPolling_IT()*
- *HAL_XSPI_MemoryMapped()*
- *HAL_XSPI_ErrorCallback()*
- *HAL_XSPI_AbortCpltCallback()*
- *HAL_XSPI_FifoThresholdCallback()*
- *HAL_XSPI_CmdCpltCallback()*
- *HAL_XSPI_RxCpltCallback()*
- *HAL_XSPI_TxCpltCallback()*
- *HAL_XSPI_RxHalfCpltCallback()*
- *HAL_XSPI_TxHalfCpltCallback()*
- *HAL_XSPI_StatusMatchCallback()*
- *HAL_XSPI_TimeOutCallback()*
- *HAL_XSPI_RegisterCallback()*
- *HAL_XSPI_UnRegisterCallback()*

89.2.4 Peripheral Control and State functions

This subsection provides a set of functions allowing to :
- Check in run-time the state of the driver.
- Check the error code set during last operation.
- Abort any operation.
- Manage the Fifo threshold.
- Configure the timeout duration used in the driver.

This section contains the following APIs:
- *HAL_XSPI_Abort()*
- *HAL_XSPI_Abort_IT()*
- *HAL_XSPI_SetFifoThreshold()*
- *HAL_XSPI_GetFifoThreshold()*
- *HAL_XSPI_SetMemoryType()*
- *HAL_XSPI_SetDeviceSize()*
- *HAL_XSPI_SetClockPrescaler()*
- *HAL_XSPI_SetTimeout()*
- *HAL_XSPI_GetError()*
- *HAL_XSPI_GetState()*

89.2.5 IO Manager configuration function

This subsection provides a set of functions allowing to :
- Configure the IO manager.

This section contains the following APIs:
- *HAL_XSPIM_Config()*

89.2.6 Delay Block function

This subsection provides a set of functions allowing to :
- Configure the delay block.

This section contains the following APIs:
- *HAL_XSPI_DLYB_SetConfig()*
- *HAL_XSPI_DLYB_GetConfig()*
- *HAL_XSPI_DLYB_GetClockPeriod()*

89.2.7 High-speed interface and calibration functions

This subsection provides a set of functions allowing to :
- Get the delay values of the high-speed interface DLLs.
- Set a delay value for the high-speed interface DLLs.

This section contains the following APIs:
- *HAL_XSPI_GetDelayValue()*
- *HAL_XSPI_SetDelayValue()*

89.2.8 Detailed description of functions

HAL_XSPI_Init

Function name

HAL_StatusTypeDef HAL_XSPI_Init (XSPI_HandleTypeDef * hxspi)

Function description

Initialize the XSPI mode according to the specified parameters in the XSPI_InitTypeDef and initialize the associated handle.

Parameters
- **hxspi:** : XSPI handle

Return values
- **HAL:** status

HAL_XSPI_MspInit

Function name

void HAL_XSPI_MspInit (XSPI_HandleTypeDef * hxspi)

Function description

Initialize the XSPI MSP.

Parameters
- **hxspi:** : XSPI handle

Return values
- **None:**

HAL_XSPI_DeInit

Function name

HAL_StatusTypeDef HAL_XSPI_DeInit (XSPI_HandleTypeDef * hxspi)

Function description

De-Initialize the XSPI peripheral.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_MspDeInit

Function name

void HAL_XSPI_MspDeInit (XSPI_HandleTypeDef * hxspi)

Function description

DeInitialize the XSPI MSP.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_IRQHandler

Function name

void HAL_XSPI_IRQHandler (XSPI_HandleTypeDef * hxspi)

Function description

Handle XSPI interrupt request.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_Command

Function name

HAL_StatusTypeDef HAL_XSPI_Command (XSPI_HandleTypeDef * hxspi, XSPI_RegularCmdTypeDef *const pCmd, uint32_t Timeout)

Function description

Set the command configuration.

Parameters

- **hxspi:** : XSPI handle
- **pCmd:** : structure that contains the command configuration information
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_Command_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Command_IT (XSPI_HandleTypeDef * hxspi, XSPI_RegularCmdTypeDef *const pCmd)

Function description

Set the command configuration in interrupt mode.

Parameters

- **hxspi:** : XSPI handle
- **pCmd:** : structure that contains the command configuration information

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read or Write Modes

HAL_XSPI_HyperbusCfg

Function name

HAL_StatusTypeDef HAL_XSPI_HyperbusCfg (XSPI_HandleTypeDef * hxspi, XSPI_HyperbusCfgTypeDef *const pCfg, uint32_t Timeout)

Function description

Configure the Hyperbus parameters.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to Structure containing the Hyperbus configuration
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_HyperbusCmd

Function name

HAL_StatusTypeDef HAL_XSPI_HyperbusCmd (XSPI_HandleTypeDef * hxspi, XSPI_HyperbusCmdTypeDef *const pCmd, uint32_t Timeout)

Function description

Set the Hyperbus command configuration.

Parameters

- **hxspi:** : XSPI handle
- **pCmd:** : Structure containing the Hyperbus command
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_Transmit

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit (XSPI_HandleTypeDef * hxspi, uint8_t *const pData, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_XSPI_Receive

Function name

HAL_StatusTypeDef HAL_XSPI_Receive (XSPI_HandleTypeDef * hxspi, uint8_t *const pData, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_XSPI_Transmit_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit_IT (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Send an amount of data in non-blocking mode with interrupt.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode

HAL_XSPI_Receive_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Receive_IT (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Receive an amount of data in non-blocking mode with interrupt.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode

HAL_XSPI_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_XSPI_Transmit_DMA (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Send an amount of data in non-blocking mode with DMA.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Write Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_XSPI_Receive_DMA

Function name

HAL_StatusTypeDef HAL_XSPI_Receive_DMA (XSPI_HandleTypeDef * hxspi, uint8_t *const pData)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hxspi:** : XSPI handle
- **pData:** : pointer to data buffer.

Return values

- **HAL:** status

Notes

- This function is used only in Indirect Read Mode
- If DMA peripheral access is configured as halfword, the number of data and the fifo threshold should be aligned on halfword
- If DMA peripheral access is configured as word, the number of data and the fifo threshold should be aligned on word

HAL_XSPI_AutoPolling

Function name

HAL_StatusTypeDef HAL_XSPI_AutoPolling (XSPI_HandleTypeDef * hxspi, XSPI_AutoPollingTypeDef *const pCfg, uint32_t Timeout)

Function description

Configure the XSPI Automatic Polling Mode in blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to structure that contains the polling configuration information.
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_XSPI_AutoPolling_IT

Function name

HAL_StatusTypeDef HAL_XSPI_AutoPolling_IT (XSPI_HandleTypeDef * hxspi, XSPI_AutoPollingTypeDef *const pCfg)

Function description

Configure the XSPI Automatic Polling Mode in non-blocking mode.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to structure that contains the polling configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Automatic Polling Mode

HAL_XSPI_MemoryMapped

Function name

HAL_StatusTypeDef HAL_XSPI_MemoryMapped (XSPI_HandleTypeDef * hxspi, XSPI_MemoryMappedTypeDef *const pCfg)

Function description

Configure the Memory Mapped mode.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to structure that contains the memory mapped configuration information.

Return values

- **HAL:** status

Notes

- This function is used only in Memory mapped Mode

HAL_XSPI_ErrorCallback

Function name

void HAL_XSPI_ErrorCallback (XSPI_HandleTypeDef * hxspi)

Function description

Transfer Error callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_AbortCpltCallback

Function name

void HAL_XSPI_AbortCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Abort completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_FifoThresholdCallback

Function name

void HAL_XSPI_FifoThresholdCallback (XSPI_HandleTypeDef * hxspi)

Function description

FIFO Threshold callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_CmdCpltCallback

Function name

void HAL_XSPI_CmdCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Command completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_RxCpltCallback

Function name

void HAL_XSPI_RxCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Rx Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_TxCpltCallback

Function name

void HAL_XSPI_TxCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Tx Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_RxHalfCpltCallback

Function name

void HAL_XSPI_RxHalfCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Rx Half Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_TxHalfCpltCallback

Function name

void HAL_XSPI_TxHalfCpltCallback (XSPI_HandleTypeDef * hxspi)

Function description

Tx Half Transfer completed callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_StatusMatchCallback

Function name

void HAL_XSPI_StatusMatchCallback (XSPI_HandleTypeDef * hxspi)

Function description

Status Match callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_TimeOutCallback

Function name

void HAL_XSPI_TimeOutCallback (XSPI_HandleTypeDef * hxspi)

Function description

Timeout callback.

Parameters

- **hxspi:** : XSPI handle

Return values

- **None:**

HAL_XSPI_RegisterCallback

Function name

HAL_StatusTypeDef HAL_XSPI_RegisterCallback (XSPI_HandleTypeDef * hxspi, HAL_XSPI_CallbackIDTypeDef CallbackID, pXSPI_CallbackTypeDef pCallback)

Function description

Register a User XSPI Callback To be used to override the weak predefined callback.

Parameters

- **hxspi:** : XSPI handle
- **CallbackID:** : ID of the callback to be registered This parameter can be one of the following values:
  – HAL_XSPI_ERROR_CB_ID XSPI Error Callback ID
  – HAL_XSPI_ABORT_CB_ID XSPI Abort Callback ID
  – HAL_XSPI_FIFO_THRESHOLD_CB_ID XSPI FIFO Threshold Callback ID
  – HAL_XSPI_CMD_CPLT_CB_ID XSPI Command Complete Callback ID
  – HAL_XSPI_RX_CPLT_CB_ID XSPI Rx Complete Callback ID
  – HAL_XSPI_TX_CPLT_CB_ID XSPI Tx Complete Callback ID
  – HAL_XSPI_RX_HALF_CPLT_CB_ID XSPI Rx Half Complete Callback ID
  – HAL_XSPI_TX_HALF_CPLT_CB_ID XSPI Tx Half Complete Callback ID
  – HAL_XSPI_STATUS_MATCH_CB_ID XSPI Status Match Callback ID
  – HAL_XSPI_TIMEOUT_CB_ID XSPI Timeout Callback ID
  – HAL_XSPI_MSP_INIT_CB_ID XSPI MspInit callback ID
  – HAL_XSPI_MSP_DEINIT_CB_ID XSPI MspDeInit callback ID
- **pCallback:** : pointer to the Callback function

Return values

- **status:**

HAL_XSPI_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_XSPI_UnRegisterCallback (XSPI_HandleTypeDef * hxspi, HAL_XSPI_CallbackIDTypeDef CallbackID)

Function description

Unregister a User XSPI Callback XSPI Callback is redirected to the weak predefined callback.

Parameters

- **hxspi:** : XSPI handle
- **CallbackID:** : ID of the callback to be unregistered This parameter can be one of the following values:
  – HAL_XSPI_ERROR_CB_ID XSPI Error Callback ID
  – HAL_XSPI_ABORT_CB_ID XSPI Abort Callback ID
  – HAL_XSPI_FIFO_THRESHOLD_CB_ID XSPI FIFO Threshold Callback ID
  – HAL_XSPI_CMD_CPLT_CB_ID XSPI Command Complete Callback ID
  – HAL_XSPI_RX_CPLT_CB_ID XSPI Rx Complete Callback ID
  – HAL_XSPI_TX_CPLT_CB_ID XSPI Tx Complete Callback ID
  – HAL_XSPI_RX_HALF_CPLT_CB_ID XSPI Rx Half Complete Callback ID
  – HAL_XSPI_TX_HALF_CPLT_CB_ID XSPI Tx Half Complete Callback ID
  – HAL_XSPI_STATUS_MATCH_CB_ID XSPI Status Match Callback ID
  – HAL_XSPI_TIMEOUT_CB_ID XSPI Timeout Callback ID
  – HAL_XSPI_MSP_INIT_CB_ID XSPI MspInit callback ID
  – HAL_XSPI_MSP_DEINIT_CB_ID XSPI MspDeInit callback ID

Return values

- **status:**

HAL_XSPI_Abort

Function name

HAL_StatusTypeDef HAL_XSPI_Abort (XSPI_HandleTypeDef * hxspi)

Function description

Abort the current operation, return to the indirect mode.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_Abort_IT

Function name

HAL_StatusTypeDef HAL_XSPI_Abort_IT (XSPI_HandleTypeDef * hxspi)

Function description

Abort the current operation, return to the indirect mode.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** status

HAL_XSPI_SetFifoThreshold

Function name

HAL_StatusTypeDef HAL_XSPI_SetFifoThreshold (XSPI_HandleTypeDef * hxspi, uint32_t Threshold)

Function description

Set XSPI Fifo threshold.

Parameters

- **hxspi:** : XSPI handle.
- **Threshold:** : Threshold of the Fifo.

Return values

- **HAL:** status

HAL_XSPI_GetFifoThreshold

Function name

uint32_t HAL_XSPI_GetFifoThreshold (const XSPI_HandleTypeDef * hxspi)

Function description

Get XSPI Fifo threshold.

Parameters

- **hxspi:** : XSPI handle.

Return values

- **Fifo:** threshold

HAL_XSPI_SetMemoryType

Function name

HAL_StatusTypeDef HAL_XSPI_SetMemoryType (XSPI_HandleTypeDef * hxspi, uint32_t Type)

Function description

Set XSPI Memory Type.

Parameters

- **hxspi:** : XSPI handle.
- **Type:** : Memory Type.

Return values

- **HAL:** status

HAL_XSPI_SetDeviceSize

Function name

HAL_StatusTypeDef HAL_XSPI_SetDeviceSize (XSPI_HandleTypeDef * hxspi, uint32_t Size)

Function description

Set XSPI Device Size.

Parameters

- **hxspi:** : XSPI handle.
- **Size:** : Device Size.

Return values

- **HAL:** status

HAL_XSPI_SetClockPrescaler

Function name

HAL_StatusTypeDef HAL_XSPI_SetClockPrescaler (XSPI_HandleTypeDef * hxspi, uint32_t Prescaler)

Function description

Set XSPI Clock prescaler.

Parameters

- **hxspi:** : XSPI handle.
- **Prescaler:** : Clock prescaler.

Return values

- **HAL:** status

HAL_XSPI_SetTimeout

Function name

HAL_StatusTypeDef HAL_XSPI_SetTimeout (XSPI_HandleTypeDef * hxspi, uint32_t Timeout)

Function description

Set XSPI timeout.

Parameters

- **hxspi:** : XSPI handle.
- **Timeout:** : Timeout for the memory access.

Return values

- **HAL:** state

HAL_XSPI_GetError

Function name

uint32_t HAL_XSPI_GetError (const XSPI_HandleTypeDef * hxspi)

Function description

Return the XSPI error code.

Parameters

- **hxspi:** : XSPI handle

Return values

- **XSPI:** Error Code

HAL_XSPI_GetState

Function name

uint32_t HAL_XSPI_GetState (const XSPI_HandleTypeDef * hxspi)

Function description

Return the XSPI handle state.

Parameters

- **hxspi:** : XSPI handle

Return values

- **HAL:** state

HAL_XSPIM_Config

Function name

HAL_StatusTypeDef HAL_XSPIM_Config (XSPI_HandleTypeDef *const hxspi, XSPIM_CfgTypeDef *const pCfg, uint32_t Timeout)

Function description

Configure the XSPI IO manager.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Pointer to Configuration of the IO Manager for the instance
- **Timeout:** : Timeout duration

Return values

- **HAL:** status

HAL_XSPI_DLYB_SetConfig

Function name

HAL_StatusTypeDef HAL_XSPI_DLYB_SetConfig (XSPI_HandleTypeDef * hxspi, HAL_XSPI_DLYB_CfgTypeDef *const pdlyb_cfg)

Function description

Set the Delay Block configuration.

Parameters

- **hxspi:** : XSPI handle.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **HAL:** status.

HAL_XSPI_DLYB_GetConfig

Function name

HAL_StatusTypeDef HAL_XSPI_DLYB_GetConfig (XSPI_HandleTypeDef * hxspi, HAL_XSPI_DLYB_CfgTypeDef *const pdlyb_cfg)

Function description

Get the Delay Block configuration.

Parameters

- **hxspi:** : XSPI handle.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **HAL:** status.

HAL_XSPI_DLYB_GetClockPeriod

Function name

HAL_StatusTypeDef HAL_XSPI_DLYB_GetClockPeriod (XSPI_HandleTypeDef * hxspi, HAL_XSPI_DLYB_CfgTypeDef *const pdlyb_cfg)

Function description

Get the Delay line length value.

Parameters

- **hxspi:** : XSPI handle.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **HAL:** status.

HAL_XSPI_GetDelayValue

Function name

HAL_StatusTypeDef HAL_XSPI_GetDelayValue (XSPI_HandleTypeDef * hxspi, XSPI_HSCalTypeDef *const pCfg)

Function description

Get the delay values of the high-speed interface DLLs.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Current delay values corresponding to the DelayValueType field.

Return values

- **HAL:** status

HAL_XSPI_SetDelayValue

Function name

HAL_StatusTypeDef HAL_XSPI_SetDelayValue (XSPI_HandleTypeDef * hxspi, XSPI_HSCalTypeDef *const pCfg)

Function description

Set a delay value for the high-speed interface DLLs.

Parameters

- **hxspi:** : XSPI handle
- **pCfg:** : Configuration of delay value specified in DelayValueType field.

Return values

- **HAL:** status

89.3 XSPI Firmware driver defines

The following section lists the various define and macros of the module.

89.3.1 XSPI

XSPI

XSPI Address DTR Mode

HAL_XSPI_ADDRESS_DTR_DISABLE

DTR mode disabled for address phase

HAL_XSPI_ADDRESS_DTR_ENABLE

DTR mode enabled for address phase

XSPI Address Mode

HAL_XSPI_ADDRESS_NONE

No address

HAL_XSPI_ADDRESS_1_LINE

Address on a single line

HAL_XSPI_ADDRESS_2_LINES

Address on two lines

HAL_XSPI_ADDRESS_4_LINES

Address on four lines

HAL_XSPI_ADDRESS_8_LINES

Address on eight lines

XSPI Hyperbus Address Space

HAL_XSPI_MEMORY_ADDRESS_SPACE

HyperBus memory mode

HAL_XSPI_REGISTER_ADDRESS_SPACE

HyperBus register mode

XSPI Address width

HAL_XSPI_ADDRESS_8_BITS

8-bit address

HAL_XSPI_ADDRESS_16_BITS

16-bit address

HAL_XSPI_ADDRESS_24_BITS

24-bit address

HAL_XSPI_ADDRESS_32_BITS

32-bit address

XSPI Alternate Bytes DTR Mode

HAL_XSPI_ALT_BYTES_DTR_DISABLE

DTR mode disabled for alternate bytes phase

HAL_XSPI_ALT_BYTES_DTR_ENABLE

DTR mode enabled for alternate bytes phase

XSPI Alternate Bytes Mode

HAL_XSPI_ALT_BYTES_NONE

No alternate bytes

HAL_XSPI_ALT_BYTES_1_LINE

Alternate bytes on a single line

HAL_XSPI_ALT_BYTES_2_LINES

Alternate bytes on two lines

HAL_XSPI_ALT_BYTES_4_LINES

Alternate bytes on four lines

HAL_XSPI_ALT_BYTES_8_LINES

Alternate bytes on eight lines

XSPI Alternate Bytes Width

HAL_XSPI_ALT_BYTES_8_BITS

8-bit alternate bytes

HAL_XSPI_ALT_BYTES_16_BITS

16-bit alternate bytes

HAL_XSPI_ALT_BYTES_24_BITS

24-bit alternate bytes

HAL_XSPI_ALT_BYTES_32_BITS

32-bit alternate bytes

XSPI Automatic Stop

HAL_XSPI_AUTOMATIC_STOP_DISABLE

AutoPolling stops only with abort or XSPI disabling

HAL_XSPI_AUTOMATIC_STOP_ENABLE

AutoPolling stops as soon as there is a match

XSPI Chip Select Boundary

HAL_XSPI_BONDARYOF_NONE

HAL_XSPI_BONDARYOF_16B
16 bits (2 Byte = 2^(1))

HAL_XSPI_BONDARYOF_32B
32 bits (4 Byte = 2^(2))

HAL_XSPI_BONDARYOF_64B
64 bits (8 Byte = 2^(3))

HAL_XSPI_BONDARYOF_128B
128 bits (16 Byte = 2^(4))

HAL_XSPI_BONDARYOF_256B
256 bits (32 Byte = 2^(5))

HAL_XSPI_BONDARYOF_512B
512 bits (64 Byte = 2^(6))

HAL_XSPI_BONDARYOF_1KB
1 Kbits (128 Byte = 2^(7))

HAL_XSPI_BONDARYOF_2KB
2 Kbits (256 Byte = 2^(8))

HAL_XSPI_BONDARYOF_4KB
4 Kbits (512 Byte = 2^(9))

HAL_XSPI_BONDARYOF_8KB
8 Kbits (1 KByte = 2^(10))

HAL_XSPI_BONDARYOF_16KB
16 Kbits (2 KByte = 2^(11))

HAL_XSPI_BONDARYOF_32KB
32 Kbits (4 KByte = 2^(12))

HAL_XSPI_BONDARYOF_64KB
64 Kbits (8 KByte = 2^(13))

HAL_XSPI_BONDARYOF_128KB
128 Kbits (16 KByte = 2^(14))

HAL_XSPI_BONDARYOF_256KB
256 Kbits (32 KByte = 2^(15))

HAL_XSPI_BONDARYOF_512KB
512 Kbits (64 KByte = 2^(16))

HAL_XSPI_BONDARYOF_1MB
1 Mbits (128 KByte = 2^(17))

HAL_XSPI_BONDARYOF_2MB
2 Mbits (256 KByte = 2^(18))

HAL_XSPI_BONDARYOF_4MB
4 Mbits (512 KByte = 2^(19))

HAL_XSPI_BONDARYOF_8MB
8 Mbits (1 MByte = 2^(20))

HAL_XSPI_BONDARYOF_16MB
16 Mbits (2 MByte = 2^(21))

HAL_XSPI_BONDARYOF_32MB
32 Mbits (4 MByte = 2^(22))

HAL_XSPI_BONDARYOF_64MB
64 Mbits (8 MByte = 2^(23))

HAL_XSPI_BONDARYOF_128MB
128 Mbits (16 MByte = 2^(24))

HAL_XSPI_BONDARYOF_256MB
256 Mbits (32 MByte = 2^(25))

HAL_XSPI_BONDARYOF_512MB
512 Mbits (64 MByte = 2^(26))

HAL_XSPI_BONDARYOF_1GB
1 Gbits (128 MByte = 2^(27))

HAL_XSPI_BONDARYOF_2GB
2 Gbits (256 MByte = 2^(28))

HAL_XSPI_BONDARYOF_4GB
4 Gbits (512 MByte = 2^(29))

HAL_XSPI_BONDARYOF_8GB
8 Gbits (1 GByte = 2^(30))

HAL_XSPI_BONDARYOF_16GB
16 Gbits (2 GByte = 2^(31))

XSPI Clock Mode

HAL_XSPI_CLOCK_MODE_0
CLK must stay low while nCS is high

HAL_XSPI_CLOCK_MODE_3
CLK must stay high while nCS is high

XSPI Data DTR Mode

HAL_XSPI_DATA_DTR_DISABLE
DTR mode disabled for data phase

HAL_XSPI_DATA_DTR_ENABLE
DTR mode enabled for data phase

XSPI Data Mode

HAL_XSPI_DATA_NONE
No data

HAL_XSPI_DATA_1_LINE

Data on a single line

HAL_XSPI_DATA_2_LINES

Data on two lines

HAL_XSPI_DATA_4_LINES

Data on four lines

HAL_XSPI_DATA_8_LINES

Data on eight lines

HAL_XSPI_DATA_16_LINES

Data on sixteen lines valid for HSPI only

XSPI Delay Block Bypaas

HAL_XSPI_DELAY_BLOCK_ON

Sampling clock is delayed by the delay block

HAL_XSPI_DELAY_BLOCK_BYPASS

Delay block is bypassed

XSPI Delay Hold Quarter Cycle

HAL_XSPI_DHQC_DISABLE

No Delay

HAL_XSPI_DHQC_ENABLE

Delay Hold 1/4 cycle

XSPI Calibration Delay Type

HAL_XSPI_CAL_FULL_CYCLE_DELAY

Delay value equivalent to full memory-clock cycle

HAL_XSPI_CAL_FEEDBACK_CLK_DELAY

Delay value for the feedback clock when reading without DQS

HAL_XSPI_CAL_DATA_OUTPUT_DELAY

Delay value for output data in DDR mode for write operations

HAL_XSPI_CAL_DQS_INPUT_DELAY

Delay value for DQS input when sampling data for read operations

XSPI DQS Mode

HAL_XSPI_DQS_DISABLE

DQS disabled

HAL_XSPI_DQS_ENABLE

DQS enabled

XSPI Error Code

HAL_XSPI_ERROR_NONE

No error

HAL_XSPI_ERROR_TIMEOUT

Timeout error

HAL_XSPI_ERROR_TRANSFER

Transfer error

HAL_XSPI_ERROR_DMA

DMA transfer error

HAL_XSPI_ERROR_INVALID_PARAM

Invalid parameters error

HAL_XSPI_ERROR_INVALID_SEQUENCE

Sequence is incorrect

HAL_XSPI_ERROR_INVALID_CALLBACK

Invalid callback error

XSPI Exported Macros

HAL_XSPI_RESET_HANDLE_STATE

Description:

- Reset XSPI handle state.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_ENABLE

Description:

- Enable the XSPI peripheral.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_DISABLE

Description:

- Disable the XSPI peripheral.

Parameters:

- __HANDLE__: specifies the XSPI Handle.

Return value:

- None

HAL_XSPI_ENABLE_IT

Description:
- Enable the specified XSPI interrupt.

Parameters:
- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to enable. This parameter can be one of the following values:
  – HAL_XSPI_IT_TO: XSPI Timeout interrupt
  – HAL_XSPI_IT_SM: XSPI Status match interrupt
  – HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  – HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  – HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:
- None

HAL_XSPI_DISABLE_IT

Description:
- Disable the specified XSPI interrupt.

Parameters:
- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to disable. This parameter can be one of the following values:
  – HAL_XSPI_IT_TO: XSPI Timeout interrupt
  – HAL_XSPI_IT_SM: XSPI Status match interrupt
  – HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  – HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  – HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:
- None

HAL_XSPI_GET_IT_SOURCE

Description:
- Check whether the specified XSPI interrupt source is enabled or not.

Parameters:
- __HANDLE__: specifies the XSPI Handle.
- __INTERRUPT__: specifies the XSPI interrupt source to check. This parameter can be one of the following values:
  – HAL_XSPI_IT_TO: XSPI Timeout interrupt
  – HAL_XSPI_IT_SM: XSPI Status match interrupt
  – HAL_XSPI_IT_FT: XSPI FIFO threshold interrupt
  – HAL_XSPI_IT_TC: XSPI Transfer complete interrupt
  – HAL_XSPI_IT_TE: XSPI Transfer error interrupt

Return value:
- The: new state of __INTERRUPT__ (TRUE or FALSE).

HAL_XSPI_GET_FLAG

Description:

- Check whether the selected XSPI flag is set or not.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __FLAG__: specifies the XSPI flag to check. This parameter can be one of the following values:
  - HAL_XSPI_FLAG_BUSY: XSPI Busy flag
  - HAL_XSPI_FLAG_TO: XSPI Timeout flag
  - HAL_XSPI_FLAG_SM: XSPI Status match flag
  - HAL_XSPI_FLAG_FT: XSPI FIFO threshold flag
  - HAL_XSPI_FLAG_TC: XSPI Transfer complete flag
  - HAL_XSPI_FLAG_TE: XSPI Transfer error flag

Return value:

- None

HAL_XSPI_CLEAR_FLAG

Description:

- Clears the specified XSPI's flag status.

Parameters:

- __HANDLE__: specifies the XSPI Handle.
- __FLAG__: specifies the XSPI clear register flag that needs to be set This parameter can be one of the following values:
  - HAL_XSPI_FLAG_TO: XSPI Timeout flag
  - HAL_XSPI_FLAG_SM: XSPI Status match flag
  - HAL_XSPI_FLAG_TC: XSPI Transfer complete flag
  - HAL_XSPI_FLAG_TE: XSPI Transfer error flag

Return value:

- None

XSPI Exported Types

HAL_XSPI_DLYB_CfgTypeDef

XSPI Flags

HAL_XSPI_FLAG_BUSY

Busy flag: operation is ongoing

HAL_XSPI_FLAG_TO

Timeout flag: timeout occurs in memory-mapped mode

HAL_XSPI_FLAG_SM

Status match flag: received data matches in autopolling mode

HAL_XSPI_FLAG_FT

Fifo threshold flag: Fifo threshold reached or data left after read from memory is complete

HAL_XSPI_FLAG_TC

Transfer complete flag: programmed number of data have been transferred or the transfer has been aborted

HAL_XSPI_FLAG_TE

Transfer error flag: invalid address is being accessed

XSPI Free Running Clock

HAL_XSPI_FREERUNCLK_DISABLE

CLK is not free running

HAL_XSPI_FREERUNCLK_ENABLE

CLK is always provided (running)

XSPI Instruction DTR Mode

HAL_XSPI_INSTRUCTION_DTR_DISABLE

DTR mode disabled for instruction phase

HAL_XSPI_INSTRUCTION_DTR_ENABLE

DTR mode enabled for instruction phase

XSPI Instruction Mode

HAL_XSPI_INSTRUCTION_NONE

No instruction

HAL_XSPI_INSTRUCTION_1_LINE

Instruction on a single line

HAL_XSPI_INSTRUCTION_2_LINES

Instruction on two lines

HAL_XSPI_INSTRUCTION_4_LINES

Instruction on four lines

HAL_XSPI_INSTRUCTION_8_LINES

Instruction on eight lines

XSPI Instruction Width

HAL_XSPI_INSTRUCTION_8_BITS

8-bit instruction

HAL_XSPI_INSTRUCTION_16_BITS

16-bit instruction

HAL_XSPI_INSTRUCTION_24_BITS

24-bit instruction

HAL_XSPI_INSTRUCTION_32_BITS

32-bit instruction

XSPI Interrupts

HAL_XSPI_IT_TO

Interrupt on the timeout flag

HAL_XSPI_IT_SM

Interrupt on the status match flag

HAL_XSPI_IT_FT

Interrupt on the fifo threshold flag

HAL_XSPI_IT_TC

Interrupt on the transfer complete flag

HAL_XSPI_IT_TE

Interrupt on the transfer error flag

XSPI IO Select

HAL_XSPI_SELECT_IO_3_0

Data exchanged over IO[3:0]

HAL_XSPI_SELECT_IO_7_4

Data exchanged over IO[7:4]

HAL_XSPI_SELECT_IO_11_8

Data exchanged over IO[11:8]

HAL_XSPI_SELECT_IO_15_12

Data exchanged over IO[15:12]

HAL_XSPI_SELECT_IO_7_0

Data exchanged over IO[7:0]

HAL_XSPI_SELECT_IO_15_8

Data exchanged over IO[15:8]

XSPI IO Port

HAL_XSPIM_IOPORT_NONE

IOs not used

HAL_XSPIM_IOPORT_1_LOW

Port 1 - IO[3:0]

HAL_XSPIM_IOPORT_1_HIGH

Port 1 - IO[7:4]

HAL_XSPIM_IOPORT_2_LOW

Port 2 - IO[3:0]

HAL_XSPIM_IOPORT_2_HIGH

Port 2 - IO[7:4]

HAL_XSPIM_IOPORT_3_LOW

Port 3 - IO[3:0]

HAL_XSPIM_IOPORT_3_HIGH

Port 3 - IO[7:4]

HAL_XSPIM_IOPORT_4_LOW

Port 4 - IO[3:0]

HAL_XSPIM_IOPORT_4_HIGH

Port 4 - IO[7:4]

HAL_XSPIM_IOPORT_5_LOW

Port 5 - IO[3:0]

HAL_XSPIM_IOPORT_5_HIGH

Port 5 - IO[7:4]

HAL_XSPIM_IOPORT_6_LOW

Port 6 - IO[3:0]

HAL_XSPIM_IOPORT_6_HIGH

Port 6 - IO[7:4]

HAL_XSPIM_IOPORT_7_LOW

Port 7 - IO[3:0]

HAL_XSPIM_IOPORT_7_HIGH

Port 7 - IO[7:4]

HAL_XSPIM_IOPORT_8_LOW

Port 8 - IO[3:0]

HAL_XSPIM_IOPORT_8_HIGH

Port 8 - IO[7:4]

XSPI Hyperbus Latency Mode

HAL_XSPI_VARIABLE_LATENCY

Variable initial latency

HAL_XSPI_FIXED_LATENCY

Fixed latency

XSPI Match Mode

HAL_XSPI_MATCH_MODE_AND

AND match mode between unmasked bits

HAL_XSPI_MATCH_MODE_OR

OR match mode between unmasked bits

XSPI Calibration Maximal Value

HAL_XSPI_MAXCAL_NOT_REACHED

Memory-clock perido inside the range of DLL master

HAL_XSPI_MAXCAL_REACHED

Memory-clock period outside the range of DLL master (max delay values used)

XSPI Memory Mode

HAL_XSPI_SINGLE_MEM

Dual-memory mode disabled

HAL_XSPI_DUAL_MEM

Dual mode enabled

XSPI Memory Size

HAL_XSPI_SIZE_16B

16 bits (2 Byte = 2^(0+1))

HAL_XSPI_SIZE_32B
32 bits (4 Byte = 2^(1+1))

HAL_XSPI_SIZE_64B
64 bits (8 Byte = 2^(2+1))

HAL_XSPI_SIZE_128B
128 bits (16 Byte = 2^(3+1))

HAL_XSPI_SIZE_256B
256 bits (32 Byte = 2^(4+1))

HAL_XSPI_SIZE_512B
512 bits (64 Byte = 2^(5+1))

HAL_XSPI_SIZE_1KB
1 Kbits (128 Byte = 2^(6+1))

HAL_XSPI_SIZE_2KB
2 Kbits (256 Byte = 2^(7+1))

HAL_XSPI_SIZE_4KB
4 Kbits (512 Byte = 2^(8+1))

HAL_XSPI_SIZE_8KB
8 Kbits (1 KByte = 2^(9+1))

HAL_XSPI_SIZE_16KB
16 Kbits (2 KByte = 2^(10+1))

HAL_XSPI_SIZE_32KB
32 Kbits (4 KByte = 2^(11+1))

HAL_XSPI_SIZE_64KB
64 Kbits (8 KByte = 2^(12+1))

HAL_XSPI_SIZE_128KB
128 Kbits (16 KByte = 2^(13+1))

HAL_XSPI_SIZE_256KB
256 Kbits (32 KByte = 2^(14+1))

HAL_XSPI_SIZE_512KB
512 Kbits (64 KByte = 2^(15+1))

HAL_XSPI_SIZE_1MB
1 Mbits (128 KByte = 2^(16+1))

HAL_XSPI_SIZE_2MB
2 Mbits (256 KByte = 2^(17+1))

HAL_XSPI_SIZE_4MB
4 Mbits (512 KByte = 2^(18+1))

HAL_XSPI_SIZE_8MB
8 Mbits (1 MByte = 2^(19+1))

HAL_XSPI_SIZE_16MB
16 Mbits (2 MByte = 2^(20+1))

HAL_XSPI_SIZE_32MB
32 Mbits (4 MByte = 2^(21+1))

HAL_XSPI_SIZE_64MB
64 Mbits (8 MByte = 2^(22+1))

HAL_XSPI_SIZE_128MB
128 Mbits (16 MByte = 2^(23+1))

HAL_XSPI_SIZE_256MB
256 Mbits (32 MByte = 2^(24+1))

HAL_XSPI_SIZE_512MB
512 Mbits (64 MByte = 2^(25+1))

HAL_XSPI_SIZE_1GB
1 Gbits (128 MByte = 2^(26+1))

HAL_XSPI_SIZE_2GB
2 Gbits (256 MByte = 2^(27+1))

HAL_XSPI_SIZE_4GB
4 Gbits (256 MByte = 2^(28+1))

HAL_XSPI_SIZE_8GB
8 Gbits (256 MByte = 2^(29+1))

HAL_XSPI_SIZE_16GB
16 Gbits (256 MByte = 2^(30+1))

HAL_XSPI_SIZE_32GB
32 Gbits (256 MByte = 2^(31+1))

XSPI Memory Type

HAL_XSPI_MEMTYPE_MICRON
Micron mode

HAL_XSPI_MEMTYPE_MACRONIX
Macronix mode

HAL_XSPI_MEMTYPE_APMEM
AP Memory mode

HAL_XSPI_MEMTYPE_MACRONIX_RAM
Macronix RAM mode

HAL_XSPI_MEMTYPE_HYPERBUS
Hyperbus mode

HAL_XSPI_MEMTYPE_APMEM_16BITS
AP Memory mode

XSPI Operation Type

HAL_XSPI_OPTYPE_COMMON_CFG

Common configuration (indirect or auto-polling mode)

HAL_XSPI_OPTYPE_READ_CFG

Read configuration (memory-mapped mode)

HAL_XSPI_OPTYPE_WRITE_CFG

Write configuration (memory-mapped mode)

HAL_XSPI_OPTYPE_WRAP_CFG

Wrap configuration (memory-mapped mode)

XSPI Sample Shifting

HAL_XSPI_SAMPLE_SHIFT_NONE

No shift

HAL_XSPI_SAMPLE_SHIFT_HALFCYCLE

1/2 cycle shift

XSPI SIOO Mode

HAL_XSPI_SIOO_INST_EVERY_CMD

Send instruction on every transaction

HAL_XSPI_SIOO_INST_ONLY_FIRST_CMD

Send instruction only for the first command

XSPI State

HAL_XSPI_STATE_RESET

Initial state

HAL_XSPI_STATE_READY

Driver ready to be used

HAL_XSPI_STATE_HYPERBUS_INIT

Initialization done in hyperbus mode but timing configuration not done

HAL_XSPI_STATE_CMD_CFG

Command (regular or hyperbus) configured, ready for an action

HAL_XSPI_STATE_READ_CMD_CFG

Read command configuration done, not the write command configuration

HAL_XSPI_STATE_WRITE_CMD_CFG

Write command configuration done, not the read command configuration

HAL_XSPI_STATE_BUSY_CMD

Command without data on-going

HAL_XSPI_STATE_BUSY_TX

Indirect Tx on-going

HAL_XSPI_STATE_BUSY_RX

Indirect Rx on-going

HAL_XSPI_STATE_BUSY_AUTO_POLLING

Auto-polling on-going

HAL_XSPI_STATE_BUSY_MEM_MAPPED

Memory-mapped on-going

HAL_XSPI_STATE_ABORT

Abort on-going

HAL_XSPI_STATE_ERROR

Blocking error, driver should be re-initialized

XSPI Timeout Activation

HAL_XSPI_TIMEOUT_COUNTER_DISABLE

Timeout counter disabled, nCS remains active

HAL_XSPI_TIMEOUT_COUNTER_ENABLE

Timeout counter enabled, nCS released when timeout expires

XSPI Timeout definition

HAL_XSPI_TIMEOUT_DEFAULT_VALUE

XSPI Wrap-Size

HAL_XSPI_WRAP_NOT_SUPPORTED

wrapped reads are not supported by the memory

HAL_XSPI_WRAP_16_BYTES

external memory supports wrap size of 16 bytes

HAL_XSPI_WRAP_32_BYTES

external memory supports wrap size of 32 bytes

HAL_XSPI_WRAP_64_BYTES

external memory supports wrap size of 64 bytes

HAL_XSPI_WRAP_128_BYTES

external memory supports wrap size of 128 bytes

XSPI Hyperbus Write Zero Latency Activation

HAL_XSPI_LATENCY_ON_WRITE

Latency on write accesses

HAL_XSPI_NO_LATENCY_ON_WRITE

No latency on write accesses

90 LL ADC Generic Driver

90.1 ADC Firmware driver registers structures

90.1.1 LL_ADC_CommonInitTypeDef

LL_ADC_CommonInitTypeDef is defined in the stm32u5xx_ll_adc.h

Data Fields

- ***uint32_t CommonClock***
- ***uint32_t Multimode***
- ***uint32_t MultiDMATransfer***
- ***uint32_t MultiTwoSamplingDelay***

Field Documentation

- ***uint32_t LL_ADC_CommonInitTypeDef::CommonClock***
  Set parameter common to several ADC: Clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_COMMON_CLOCK_SOURCE*** This feature can be modified afterwards using unitary function **LL_ADC_SetCommonClock()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::Multimode***
  Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances). This parameter can be a value of ***ADC_LL_EC_MULTI_MODE***This feature can be modified afterwards using unitary function **LL_ADC_SetMultimode()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiDMATransfer***
  Set ADC multimode conversion data transfer: no transfer or transfer by DMA. This parameter can be a value of ***ADC_LL_EC_MULTI_DMA_TRANSFER***This feature can be modified afterwards using unitary function **LL_ADC_SetMultiDMATransfer()**.
- ***uint32_t LL_ADC_CommonInitTypeDef::MultiTwoSamplingDelay***
  Set ADC multimode delay between 2 sampling phases. This parameter can be a value of ***ADC_LL_EC_MULTI_TWOSMP_DELAY***This feature can be modified afterwards using unitary function **LL_ADC_SetMultiTwoSamplingDelay()**.

90.1.2 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32u5xx_ll_adc.h

Data Fields

- ***uint32_t Resolution***
- ***uint32_t LeftBitShift***
- ***uint32_t DataAlignment***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Resolution***
  Set ADC resolution. This parameter can be a value of ***ADC_LL_EC_RESOLUTION*** This feature can be modified afterwards using unitary function **LL_ADC_SetResolution()**.
- ***uint32_t LL_ADC_InitTypeDef::LeftBitShift***
  Configures the left shifting applied to the final result with or without oversampling. This parameter can be a value of ***ADC_LL_EC_LEFT_BIT_SHIFT***.
- ***uint32_t LL_ADC_InitTypeDef::DataAlignment***
  Set ADC conversion data alignment. This parameter can be a value of ***ADC_LL_EC_DATA_ALIGN*** This feature can be modified afterwards using unitary function **LL_ADC_SetDataAlignment()**.
- ***uint32_t LL_ADC_InitTypeDef::LowPowerMode***
  Set ADC low power mode. This parameter can be a value of ***ADC_LL_EC_LP_MODE*** This feature can be modified afterwards using unitary function **LL_ADC_SetLowPowerMode()**.

90.1.3 LL_ADC_REG_InitTypeDef

LL_ADC_REG_InitTypeDef is defined in the stm32u5xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***uint32_t DataTransferMode***
- ***uint32_t DMATransfer***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t LL_ADC_REG_InitTypeDef::TriggerSource***
  Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***ADC_LL_EC_REG_TRIGGER_SOURCE***
  Note:
  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_REG_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetTriggerSource()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerLength***
  Set ADC group regular sequencer length. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_SCAN_LENGTH*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerLength()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::SequencerDiscont***
  Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of ***ADC_LL_EC_REG_SEQ_DISCONT_MODE***
  Note:
  – This parameter has an effect only if group regular sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_REG_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::ContinuousMode***
  Set ADC continuous conversion mode on ADC group regular, whether ADC conversions are performed in single mode (one conversion per trigger) or in continuous mode (after the first trigger, following conversions launched successively automatically). This parameter can be a value of ***ADC_LL_EC_REG_CONTINUOUS_MODE*** Note: It is not possible to enable both ADC group regular continuous mode and discontinuous mode. This feature can be modified afterwards using unitary function **LL_ADC_REG_SetContinuousMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::DataTransferMode***
  Set ADC group regular conversion data transfer mode: no transfer, transfer by DMA (Limited/Unlimited) or MDF. This parameter can be a value of ***ADC_LL_EC_REG_DATA_TRANSFER_MODE*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDataTransferMode()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::DMATransfer***
  Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode. This parameter can be a value of ***ADC_LL_EC_REG_DMA_TRANSFER*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetDMATransfer()**.
- ***uint32_t LL_ADC_REG_InitTypeDef::Overrun***
  Set ADC group regular behavior in case of overrun: data preserved or overwritten. This parameter can be a value of ***ADC_LL_EC_REG_OVR_DATA_BEHAVIOR*** This feature can be modified afterwards using unitary function **LL_ADC_REG_SetOverrun()**.

90.1.4 LL_ADC_INJ_InitTypeDef

LL_ADC_INJ_InitTypeDef is defined in the stm32u5xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***

- *uint32_t TrigAuto*

Field Documentation

- ***uint32_t LL_ADC_INJ_InitTypeDef::TriggerSource***
  Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of *ADC_LL_EC_INJ_TRIGGER_SOURCE*
  Note:
  – On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function **LL_ADC_INJ_SetTriggerEdge()**. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTriggerSource()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerLength***
  Set ADC group injected sequencer length. This parameter can be a value of *ADC_LL_EC_INJ_SEQ_SCAN_LENGTH* This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerLength()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::SequencerDiscont***
  Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks. This parameter can be a value of *ADC_LL_EC_INJ_SEQ_DISCONT_MODE*
  Note:
  – This parameter has an effect only if group injected sequencer is enabled (scan length of 2 ranks or more). This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetSequencerDiscont()**.
- ***uint32_t LL_ADC_INJ_InitTypeDef::TrigAuto***
  Set ADC group injected conversion trigger: independent or from ADC group regular. This parameter can be a value of *ADC_LL_EC_INJ_TRIG_AUTO* Note: This parameter must be set to set to independent trigger if injected trigger source is set to an external trigger. This feature can be modified afterwards using unitary function **LL_ADC_INJ_SetTrigAuto()**.

90.2 ADC Firmware driver API description

The following section lists the various functions of the ADC library.

90.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (const ADC_TypeDef * ADCx, uint32_t RegisterValue)

Function description

Function to help to configure DMA transfer from ADC: retrieve the ADC register address from ADC instance and a list of ADC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **ADCx:** ADC instance
- **RegisterValue:** This parameter can be one of the following values:
  – LL_ADC_DMA_REG_REGULAR_DATA
  – LL_ADC_DMA_REG_REGULAR_DATA_MULTI (1)

  (1) Available on devices with several ADC instances.

Return values

- **ADC:** register address

Notes

- These ADC registers are data registers: when ADC conversion data is available in ADC data registers, ADC generates a DMA transfer request.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, LL_ADC_DMA_GetRegAddr(ADC1, LL_ADC_DMA_REG_REGULAR_DATA), (uint32_t)&< array or variable >, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
- For devices with several ADC: in multimode, some devices use a different data register outside of ADC instance scope (common data register). This macro manages this register difference, only ADC instance has to be set as parameter.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_DMA_GetRegAddr
- CDR RDATA_MST LL_ADC_DMA_GetRegAddr
- CDR RDATA_SLV LL_ADC_DMA_GetRegAddr

LL_ADC_SetCommonClock

Function name

__STATIC_INLINE void LL_ADC_SetCommonClock (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t CommonClock)

Function description

Set parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **CommonClock:** This parameter can be one of the following values:
  – LL_ADC_CLOCK_ASYNC_DIV1
  – LL_ADC_CLOCK_ASYNC_DIV2
  – LL_ADC_CLOCK_ASYNC_DIV4
  – LL_ADC_CLOCK_ASYNC_DIV6
  – LL_ADC_CLOCK_ASYNC_DIV8
  – LL_ADC_CLOCK_ASYNC_DIV10
  – LL_ADC_CLOCK_ASYNC_DIV12
  – LL_ADC_CLOCK_ASYNC_DIV16
  – LL_ADC_CLOCK_ASYNC_DIV32
  – LL_ADC_CLOCK_ASYNC_DIV64
  – LL_ADC_CLOCK_ASYNC_DIV128
  – LL_ADC_CLOCK_ASYNC_DIV256

Return values

- **None:**

Notes

- On this STM32 series, if ADC group injected is used, some clock ratio constraints between ADC clock and AHB clock must be respected. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_SetCommonClock
- CCR PRESC LL_ADC_SetCommonClock

LL_ADC_GetCommonClock

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonClock (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: Clock source and prescaler.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CLOCK_ASYNC_DIV1
  - LL_ADC_CLOCK_ASYNC_DIV2
  - LL_ADC_CLOCK_ASYNC_DIV4
  - LL_ADC_CLOCK_ASYNC_DIV6
  - LL_ADC_CLOCK_ASYNC_DIV8
  - LL_ADC_CLOCK_ASYNC_DIV10
  - LL_ADC_CLOCK_ASYNC_DIV12
  - LL_ADC_CLOCK_ASYNC_DIV16
  - LL_ADC_CLOCK_ASYNC_DIV32
  - LL_ADC_CLOCK_ASYNC_DIV64
  - LL_ADC_CLOCK_ASYNC_DIV128
  - LL_ADC_CLOCK_ASYNC_DIV256

Reference Manual to LL API cross reference:

- CCR CKMODE LL_ADC_GetCommonClock
- CCR PRESC LL_ADC_GetCommonClock

LL_ADC_SetCommonPathInternalChAdd

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChAdd (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  - LL_ADC_PATH_INTERNAL_NONE
  - LL_ADC_PATH_INTERNAL_VREFINT
  - LL_ADC_PATH_INTERNAL_TEMPSENSOR
  - LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literals LL_ADC_DELAY_TEMPSENSOR_STAB_US,
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChAdd
- CCR VSENSESEL LL_ADC_SetCommonPathInternalChAdd
- CCR VBATEN LL_ADC_SetCommonPathInternalChAdd

LL_ADC_SetCommonPathInternalChRem

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalChRem (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  – LL_ADC_PATH_INTERNAL_NONE
  – LL_ADC_PATH_INTERNAL_VREFINT
  – LL_ADC_PATH_INTERNAL_TEMPSENSOR
  – LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalChRem
- CCR VSENSESEL LL_ADC_SetCommonPathInternalChRem
- CCR VBATEN LL_ADC_SetCommonPathInternalChRem

LL_ADC_SetCommonPathInternalCh

Function name

__STATIC_INLINE void LL_ADC_SetCommonPathInternalCh (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t PathInternal)

Function description

Set parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **PathInternal:** This parameter can be a combination of the following values:
  – LL_ADC_PATH_INTERNAL_NONE
  – LL_ADC_PATH_INTERNAL_VREFINT
  – LL_ADC_PATH_INTERNAL_TEMPSENSOR
  – LL_ADC_PATH_INTERNAL_VBAT

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)
- Stabilization time of measurement path to internal channel: After enabling internal paths, before starting ADC conversion, a delay is required for internal voltage reference and temperature sensor stabilization time. Refer to device datasheet. Refer to literal LL_ADC_DELAY_VREFINT_STAB_US. Refer to literal LL_ADC_DELAY_TEMPSENSOR_STAB_US.
- ADC internal channel sampling time constraint: For ADC conversion of internal channels, a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_SetCommonPathInternalCh
- CCR VSENSESEL LL_ADC_SetCommonPathInternalCh
- CCR VBATEN LL_ADC_SetCommonPathInternalCh

LL_ADC_GetCommonPathInternalCh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCommonPathInternalCh (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get parameter common to several ADC: measurement path to internal channels (VrefInt, temperature sensor, ...).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be a combination of the following values:
  – LL_ADC_PATH_INTERNAL_NONE
  – LL_ADC_PATH_INTERNAL_VREFINT
  – LL_ADC_PATH_INTERNAL_TEMPSENSOR
  – LL_ADC_PATH_INTERNAL_VBAT

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR)

Reference Manual to LL API cross reference:

- CCR VREFEN LL_ADC_GetCommonPathInternalCh
- CCR VSENSESEL LL_ADC_GetCommonPathInternalCh
- CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_SetCalibrationOffsetFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationOffsetFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff, uint32_t CalibrationFactor)

Function description

Set ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  – LL_ADC_SINGLE_ENDED
  – LL_ADC_DIFFERENTIAL_ENDED
  – LL_ADC_BOTH_SINGLE_DIFF_ENDED
- **CalibrationFactor:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration factor must be specified for each of these differential modes, if used afterwards and if the application requires their calibration). Calibration of linearity is common to both single-ended and differential modes (calibration factor can be specified only once).
- In case of setting calibration factors of both modes single ended and differential (parameter LL_ADC_BOTH_SINGLE_DIFF_ENDED): both calibration factors must be concatenated. To perform this processing, use helper macro __LL_ADC_CALIB_FACTOR_SINGLE_DIFF().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.

LL_ADC_GetCalibrationOffsetFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationOffsetFactor (ADC_TypeDef * ADCx, uint32_t SingleDiff)

Function description

Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **SingleDiff:** This parameter can be one of the following values:
  – LL_ADC_SINGLE_ENDED
  – LL_ADC_DIFFERENTIAL_ENDED

Return values

- **Value:** between Min_Data=0x0000 and Max_Data=0xFFFF

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes Calibration of linearity is common to both single-ended and differential modes

LL_ADC_SetCalibrationLinearFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationLinearFactor (ADC_TypeDef * ADCx, uint32_t LinearityWord, uint32_t CalibrationFactor)

Function description

Set ADC Linear calibration factor in the mode single-ended.

Parameters

- **ADCx:** ADC instance (on STM32U5, feature available on ADC instances: ADC1, ADC2)
- **LinearityWord:** This parameter can be one of the following values:
  – LL_ADC_CALIB_LINEARITY_INDEX1
  – LL_ADC_CALIB_LINEARITY_INDEX2
  – LL_ADC_CALIB_LINEARITY_INDEX3
  – LL_ADC_CALIB_LINEARITY_INDEX4
  – LL_ADC_CALIB_LINEARITY_INDEX5
  – LL_ADC_CALIB_LINEARITY_INDEX6
  – LL_ADC_CALIB_LINEARITY_INDEX7
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x3FFFFFFF

Return values

- **None:**

Notes

- This function is intended to set linear calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- On STM32U5, this feature is available on ADC instances: ADC1, ADC2.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled, without calibration on going, without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CALFACT2 LINCALFACT LL_ADC_SetCalibrationLinearFactor
- CALFACT2 LINCALFACT LL_ADC_SetCalibrationLinearFactor

LL_ADC_GetCalibrationLinearFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationLinearFactor (ADC_TypeDef * ADCx, uint32_t LinearityWord)

Function description

Get ADC Linear calibration factor in the mode single-ended.

Parameters

- **ADCx:** ADC instance (on STM32U5, feature available on ADC instances: ADC1, ADC2)
- **LinearityWord:** This parameter can be one of the following values:
  – LL_ADC_CALIB_LINEARITY_INDEX1
  – LL_ADC_CALIB_LINEARITY_INDEX2
  – LL_ADC_CALIB_LINEARITY_INDEX3
  – LL_ADC_CALIB_LINEARITY_INDEX4
  – LL_ADC_CALIB_LINEARITY_INDEX5
  – LL_ADC_CALIB_LINEARITY_INDEX6
  – LL_ADC_CALIB_LINEARITY_INDEX7

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3FFFFFFF

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().
- On STM32U5, this feature is available on ADC instances: ADC1, ADC2.

Reference Manual to LL API cross reference:

- CALFACT2 LINCALFACT LL_ADC_GetCalibrationLinearFactor
- CALFACT2 LINCALFACT LL_ADC_GetCalibrationLinearFactor

LL_ADC_SetResolution

Function name

__STATIC_INLINE void LL_ADC_SetResolution (ADC_TypeDef * ADCx, uint32_t Resolution)

Function description

Set ADC resolution.

Parameters

- **ADCx:** ADC instance
- **Resolution:** This parameter can be one of the following values:
  – LL_ADC_RESOLUTION_14B (1)
  – LL_ADC_RESOLUTION_12B
  – LL_ADC_RESOLUTION_10B
  – LL_ADC_RESOLUTION_8B
  – LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_SetResolution

LL_ADC_GetResolution

Function name

__STATIC_INLINE uint32_t LL_ADC_GetResolution (const ADC_TypeDef * ADCx)

Function description

Get ADC resolution.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_RESOLUTION_14B (1)
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4

Reference Manual to LL API cross reference:

- CFGR RES LL_ADC_GetResolution

LL_ADC_SetDataAlignment

Function name

__STATIC_INLINE void LL_ADC_SetDataAlignment (ADC_TypeDef * ADCx, uint32_t DataAlignment)

Function description

Set ADC conversion data alignment.

Parameters

- **ADCx:** ADC instance
- **DataAlignment:** This parameter can be one of the following values:
  - LL_ADC_DATA_ALIGN_RIGHT
  - LL_ADC_DATA_ALIGN_LEFT

Return values

- **None:**

Notes

- Refer to reference manual for alignments formats dependencies to ADC resolutions.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 ALIGN LL_ADC_SetDataAlignment

LL_ADC_GetDataAlignment

Function name

__STATIC_INLINE uint32_t LL_ADC_GetDataAlignment (const ADC_TypeDef * ADCx)

Function description

Get ADC conversion data alignment.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_DATA_ALIGN_RIGHT
  - LL_ADC_DATA_ALIGN_LEFT

Notes

- Refer to reference manual for alignments formats dependencies to ADC resolutions.

Reference Manual to LL API cross reference:

- CFGR1 ALIGN LL_ADC_GetDataAlignment

LL_ADC_SetLowPowerMode

Function name

__STATIC_INLINE void LL_ADC_SetLowPowerMode (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  - LL_ADC_LP_MODE_NONE
  - LL_ADC_LP_AUTOWAIT

Return values

- **None:**

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_SetLowPowerMode

LL_ADC_GetLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLowPowerMode (const ADC_TypeDef * ADCx)

Function description

Get ADC low power mode:

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_LP_MODE_NONE
  – LL_ADC_LP_AUTOWAIT

Notes

- Description of ADC low power modes: ADC low power mode "auto wait": Dynamic low power mode, ADC conversions occurrences are limited to the minimum necessary in order to reduce power consumption. New ADC conversion starts only when the previous unitary conversion data (for ADC group regular) or previous sequence conversions data (for ADC group injected) has been retrieved by user software. In the meantime, ADC remains idle: does not performs any other conversion. This mode allows to automatically adapt the ADC conversions triggers to the speed of the software that reads the data. Moreover, this avoids risk of overrun for low frequency applications. How to use this low power mode: It is not recommended to use with interruption or DMA since these modes have to clear immediately the EOC flag (by CPU to free the IRQ pending event or by DMA). Auto wait will work but fort a very short time, discarding its intended benefit (except specific case of high load of CPU or DMA transfers which can justify usage of auto wait).Do use with polling: 1. Start conversion, 2. Later on, when conversion data is needed: poll for end of conversion to ensure that conversion is completed and retrieve ADC conversion data. This will trig another ADC conversion start. ADC low power mode "auto power-off" (feature available on this device if parameter LL_ADC_LP_AUTOPOWEROFF is available): the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- With ADC low power mode "auto wait", the ADC conversion data read is corresponding to previous ADC conversion start, independently of delay during which ADC was idle. Therefore, the ADC conversion data may be outdated: does not correspond to the current voltage level on the selected ADC channel.

Reference Manual to LL API cross reference:

- CFGR AUTDLY LL_ADC_GetLowPowerMode

LL_ADC_SetOffset

Function name

__STATIC_INLINE void LL_ADC_SetOffset (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t Channel, uint32_t OffsetLevel)

Function description

Set ADC selected offset number 1, 2, 3 or 4.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- **OffsetLevel:** Value between Min_Data=0x000 and Max_Data=0x1FFFFFF

Return values

- **None:**

Notes

- This function set the 2 items of offset configuration: ADC channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)Offset level (offset to be subtracted from the raw converted data).
- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.
- This function enables the offset, by default. It can be forced to disable state using function LL_ADC_SetOffsetState().
- If a channel is mapped on several offsets numbers, only the offset with the lowest value is considered for the subtraction.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On STM32U5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_SetOffset
- OFR1 OFFSET1 LL_ADC_SetOffset
- OFR1 OFFSET1_EN LL_ADC_SetOffset
- OFR2 OFFSET2_CH LL_ADC_SetOffset
- OFR2 OFFSET2 LL_ADC_SetOffset
- OFR2 OFFSET2_EN LL_ADC_SetOffset
- OFR3 OFFSET3_CH LL_ADC_SetOffset
- OFR3 OFFSET3 LL_ADC_SetOffset
- OFR3 OFFSET3_EN LL_ADC_SetOffset
- OFR4 OFFSET4_CH LL_ADC_SetOffset
- OFR4 OFFSET4 LL_ADC_SetOffset
- OFR4 OFFSET4_EN LL_ADC_SetOffset

LL_ADC_GetOffsetChannel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetChannel (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: Channel to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  – LL_ADC_OFFSET_1
  – LL_ADC_OFFSET_2
  – LL_ADC_OFFSET_3
  – LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual). (1, 2) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().
- On STM32U5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).

Reference Manual to LL API cross reference:

- OFR1 OFFSET1_CH LL_ADC_GetOffsetChannel
- OFR2 OFFSET2_CH LL_ADC_GetOffsetChannel
- OFR3 OFFSET3_CH LL_ADC_GetOffsetChannel
- OFR4 OFFSET4_CH LL_ADC_GetOffsetChannel

LL_ADC_GetOffsetLevel

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetLevel (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: Offset level (offset to be subtracted from the raw converted data).

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x1FFFFFF

Notes

- Caution: Offset format is dependent to ADC resolution: offset has to be left-aligned on bit 11, the LSB (right bits) are set to 0.

Reference Manual to LL API cross reference:

- OFR1 OFFSET1 LL_ADC_GetOffsetLevel
- OFR2 OFFSET2 LL_ADC_GetOffsetLevel
- OFR3 OFFSET3 LL_ADC_GetOffsetLevel
- OFR4 OFFSET4 LL_ADC_GetOffsetLevel

LL_ADC_SetOffsetSign

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSign (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSign)

Function description

Set for the ADC selected offset number 1, 2, 3 or 4: choose offset sign.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetSign:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- OFR1 OFFSETPOS LL_ADC_SetOffsetSign
- OFR2 OFFSETPOS LL_ADC_SetOffsetSign
- OFR3 OFFSETPOS LL_ADC_SetOffsetSign
- OFR4 OFFSETPOS LL_ADC_SetOffsetSign

LL_ADC_GetOffsetSign

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSign (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get for the ADC selected offset number 1, 2, 3 or 4: offset sign if positive or negative.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_SIGN_NEGATIVE
  - LL_ADC_OFFSET_SIGN_POSITIVE

Reference Manual to LL API cross reference:

- OFR1 OFFSETPOS LL_ADC_GetOffsetSign
- OFR2 OFFSETPOS LL_ADC_GetOffsetSign
- OFR3 OFFSETPOS LL_ADC_GetOffsetSign
- OFR4 OFFSETPOS LL_ADC_GetOffsetSign

LL_ADC_SetOffsetSignedSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetSignedSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetSignedSaturation)

Function description

Set Signed saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  – LL_ADC_OFFSET_1
  – LL_ADC_OFFSET_2
  – LL_ADC_OFFSET_3
  – LL_ADC_OFFSET_4
- **OffsetSignedSaturation:** This parameter can be one of the following values:
  – LL_ADC_OFFSET_SIGNED_SATURATION_ENABLE
  – LL_ADC_OFFSET_SIGNED_SATURATION_DISABLE

Return values

- **Returned:** None

Reference Manual to LL API cross reference:

- OFR1 SSAT LL_ADC_SetOffsetSignedSaturation
- OFR2 SSAT LL_ADC_SetOffsetSignedSaturation
- OFR3 SSAT LL_ADC_SetOffsetSignedSaturation
- OFR4 SSAT LL_ADC_SetOffsetSignedSaturation

LL_ADC_GetOffsetSignedSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetSignedSaturation (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get Signed saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  – LL_ADC_OFFSET_1
  – LL_ADC_OFFSET_2
  – LL_ADC_OFFSET_3
  – LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_OFFSET_SIGNED_SATURATION_ENABLE
  – LL_ADC_OFFSET_SIGNED_SATURATION_DISABLE

Reference Manual to LL API cross reference:

- OFR1 SSAT LL_ADC_GetOffsetSignedSaturation
- OFR2 SSAT LL_ADC_GetOffsetSignedSaturation
- OFR3 SSAT LL_ADC_GetOffsetSignedSaturation
- OFR4 SSAT LL_ADC_GetOffsetSignedSaturation

LL_ADC_SetOffsetUnsignedSaturation

Function name

__STATIC_INLINE void LL_ADC_SetOffsetUnsignedSaturation (ADC_TypeDef * ADCx, uint32_t Offsety, uint32_t OffsetUnsignedSaturation)

Function description

Set Unsigned saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4
- **OffsetUnsignedSaturation:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_UNSIGNED_SATURATION_ENABLE
  - LL_ADC_OFFSET_UNSIGNED_SATURATION_DISABLE

Return values

- **Returned:** None

Reference Manual to LL API cross reference:

- OFR1 USAT LL_ADC_SetOffsetUnsignedSaturation
- OFR2 USAT LL_ADC_SetOffsetUnsignedSaturation
- OFR3 USAT LL_ADC_SetOffsetUnsignedSaturation
- OFR4 USAT LL_ADC_SetOffsetUnsignedSaturation

LL_ADC_GetOffsetUnsignedSaturation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOffsetUnsignedSaturation (const ADC_TypeDef * ADCx, uint32_t Offsety)

Function description

Get Unsigned saturation for the ADC selected offset number 1, 2, 3 or 4: signed offset saturation if enabled or disabled.

Parameters

- **ADCx:** ADC instance
- **Offsety:** This parameter can be one of the following values:
  - LL_ADC_OFFSET_1
  - LL_ADC_OFFSET_2
  - LL_ADC_OFFSET_3
  - LL_ADC_OFFSET_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OFFSET_UNSIGNED_SATURATION_ENABLE
  - LL_ADC_OFFSET_UNSIGNED_SATURATION_DISABLE

Reference Manual to LL API cross reference:

- OFR1 USAT LL_ADC_GetOffsetUnsignedSaturation
- OFR2 USAT LL_ADC_GetOffsetUnsignedSaturation
- OFR3 USAT LL_ADC_GetOffsetUnsignedSaturation
- OFR4 USAT LL_ADC_GetOffsetUnsignedSaturation

LL_ADC_SetGainCompensation

Function name

__STATIC_INLINE void LL_ADC_SetGainCompensation (ADC_TypeDef * ADCx, uint32_t GainCompensation)

Function description

Set ADC gain compensation.

Parameters

- **ADCx:** ADC instance
- **GainCompensation:** This parameter can be: 0 Gain compensation will be disabled and value set to 0 1 -> 16393 Gain compensation will be enabled with specified value

Return values

- **None:**

Notes

- This function set the gain compensation coefficient that is applied to raw converted data using the formula: DATA = DATA(raw) * (gain compensation coef) / 4096
- This function enables the gain compensation if given coefficient is above 0, otherwise it disables it.
- Gain compensation when enabled is applied to all channels.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- GCOMP GCOMPCOEFF LL_ADC_SetGainCompensation
- CFGR2 GCOMP LL_ADC_SetGainCompensation

LL_ADC_GetGainCompensation

Function name

__STATIC_INLINE uint32_t LL_ADC_GetGainCompensation (const ADC_TypeDef * ADCx)

Function description

Get the ADC gain compensation value.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be: 0 Gain compensation is disabled 1 -> 16393 Gain compensation is enabled with returned value

Reference Manual to LL API cross reference:

- GCOMP GCOMPCOEFF LL_ADC_GetGainCompensation
- CFGR2 GCOMP LL_ADC_GetGainCompensation

LL_ADC_SetSamplingTimeCommonChannels

Function name

__STATIC_INLINE void LL_ADC_SetSamplingTimeCommonChannels (ADC_TypeDef * ADCx, uint32_t SamplingTimeY, uint32_t SamplingTime)

Function description

Set sampling time common to a group of channels.

Parameters

- **ADCx:** ADC instance
- **SamplingTimeY:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_1
  - LL_ADC_SAMPLINGTIME_COMMON_2
- **SamplingTime:** This parameter can be one of the following values:
  - LL_ADC4_SAMPLINGTIME_1CYCLE_5
  - LL_ADC4_SAMPLINGTIME_3CYCLES_5
  - LL_ADC4_SAMPLINGTIME_7CYCLES_5
  - LL_ADC4_SAMPLINGTIME_12CYCLES_5
  - LL_ADC4_SAMPLINGTIME_19CYCLES_5
  - LL_ADC4_SAMPLINGTIME_39CYCLES_5
  - LL_ADC4_SAMPLINGTIME_79CYCLES_5
  - LL_ADC4_SAMPLINGTIME_814CYCLES_5

Return values

- **None:**

Notes

- Unit: ADC clock cycles.
- On this STM32 series, sampling time scope is on ADC instance: Sampling time common to all channels. (on some other STM32 families, sampling time is channel wise)
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- Applicable only on ADC4 instance

Reference Manual to LL API cross reference:

- SMPR SMP1 LL_ADC_SetSamplingTimeCommonChannels
- SMPR SMP2 LL_ADC_SetSamplingTimeCommonChannels

LL_ADC_GetSamplingTimeCommonChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetSamplingTimeCommonChannels (const ADC_TypeDef * ADCx, uint32_t SamplingTimeY)

Function description

Get sampling time common to a group of channels.

Parameters

- **ADCx:** ADC instance (ADC4 for this device)
- **SamplingTimeY:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_1
  - LL_ADC_SAMPLINGTIME_COMMON_2

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC4_SAMPLINGTIME_1CYCLE_5
  - LL_ADC4_SAMPLINGTIME_3CYCLES_5
  - LL_ADC4_SAMPLINGTIME_7CYCLES_5
  - LL_ADC4_SAMPLINGTIME_12CYCLES_5
  - LL_ADC4_SAMPLINGTIME_19CYCLES_5
  - LL_ADC4_SAMPLINGTIME_39CYCLES_5
  - LL_ADC4_SAMPLINGTIME_79CYCLES_5
  - LL_ADC4_SAMPLINGTIME_814CYCLES_5

Notes

- Unit: ADC clock cycles.
- On this STM32 series, sampling time scope is on ADC instance: Sampling time common to all channels. (on some other STM32 families, sampling time is channel wise)
- Conversion time is the addition of sampling time and processing time. Refer to reference manual for ADC processing time of this STM32 series.

Reference Manual to LL API cross reference:

- SMPR SMP1 LL_ADC_GetSamplingTimeCommonChannels
- SMPR SMP2 LL_ADC_GetSamplingTimeCommonChannels

LL_ADC_REG_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  – LL_ADC_REG_TRIG_SOFTWARE
  – LL_ADC_REG_TRIG_EXT_TIM1_CH1
  – LL_ADC_REG_TRIG_EXT_TIM1_CH2
  – LL_ADC_REG_TRIG_EXT_TIM1_CH3
  – LL_ADC_REG_TRIG_EXT_TIM2_CH2
  – LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM4_CH4
  – LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  – LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  – LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  – LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  – LL_ADC_REG_TRIG_EXT_TIM3_CH4
  – LL_ADC_REG_TRIG_EXT_LPTIM1_CH1
  – LL_ADC_REG_TRIG_EXT_LPTIM2_CH1
  – LL_ADC_REG_TRIG_EXT_LPTIM3_CH1

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_REG_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_SetTriggerSource
- CFGR EXTEN LL_ADC_REG_SetTriggerSource

LL_ADC_REG_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerSource (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_SOFTWARE
  - LL_ADC_REG_TRIG_EXT_TIM1_CH1
  - LL_ADC_REG_TRIG_EXT_TIM1_CH2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH3
  - LL_ADC_REG_TRIG_EXT_TIM2_CH2
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_CH4
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM4_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM6_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM15_TRGO
  - LL_ADC_REG_TRIG_EXT_TIM3_CH4
  - LL_ADC_REG_TRIG_EXT_LPTIM1_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM2_CH1
  - LL_ADC_REG_TRIG_EXT_LPTIM3_CH1

Notes

- To determine whether group regular trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_REG_GetTriggerSource(ADC1) == LL_ADC_REG_TRIG_SOFTWARE)") use function LL_ADC_REG_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CFGR EXTSEL LL_ADC_REG_GetTriggerSource
- CFGR EXTEN LL_ADC_REG_GetTriggerSource

LL_ADC_REG_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsTriggerSourceSWStart (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group regular trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_REG_GetTriggerSource().

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_IsTriggerSourceSWStart

LL_ADC_REG_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_REG_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- Applicable only for trigger source set to external trigger.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_SetTriggerEdge

LL_ADC_REG_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetTriggerEdge (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_TRIG_EXT_RISING
  - LL_ADC_REG_TRIG_EXT_FALLING
  - LL_ADC_REG_TRIG_EXT_RISINGFALLING

Notes

- Applicable only for trigger source set to external trigger.

Reference Manual to LL API cross reference:

- CFGR EXTEN LL_ADC_REG_GetTriggerEdge

LL_ADC_SetTriggerFrequencyMode

Function name

__STATIC_INLINE void LL_ADC_SetTriggerFrequencyMode (ADC_TypeDef * ADCx, uint32_t TriggerFrequencyMode)

Function description

Set ADC trigger frequency mode.

Parameters

- **ADCx:** ADC instance
- **TriggerFrequencyMode:** This parameter can be one of the following values:
  – LL_ADC_TRIGGER_FREQ_HIGH
  – LL_ADC_TRIGGER_FREQ_LOW

Return values

- **None:**

Notes

- ADC trigger frequency mode must be set to low frequency when a duration is exceeded before ADC conversion start trigger event (between ADC enable and ADC conversion start trigger event or between two ADC conversion start trigger event). Duration value: Refer to device datasheet, parameter "tIdle".
- When ADC trigger frequency mode is set to low frequency, some rearm cycles are inserted before performing ADC conversion start, inducing a delay of 2 ADC clock cycles.
- Usage of ADC trigger frequency mode with ADC low power mode: Low power mode auto wait: Only the first ADC conversion start trigger inserts the rearm delay.Low power mode auto power-off: ADC trigger frequency mode is discarded.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR2 LFTRIG LL_ADC_SetTriggerFrequencyMode

LL_ADC_GetTriggerFrequencyMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetTriggerFrequencyMode (const ADC_TypeDef * ADCx)

Function description

Get ADC trigger frequency mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_TRIGGER_FREQ_HIGH
  – LL_ADC_TRIGGER_FREQ_LOW

Reference Manual to LL API cross reference:

- CFGR2 LFTRIG LL_ADC_GetTriggerFrequencyMode

LL_ADC_REG_SetSamplingMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetSamplingMode (ADC_TypeDef * ADCx, uint32_t SamplingMode)

Function description

Set ADC sampling mode.

Parameters

- **ADCx:** ADC instance
- **SamplingMode:** This parameter can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Return values

- **None:**

Notes

- This function set the ADC conversion sampling mode
- This mode applies to regular group only.
- Set sampling mode is applied to all conversion of regular group.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_SetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_SetSamplingMode

LL_ADC_REG_GetSamplingMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSamplingMode (const ADC_TypeDef * ADCx)

Function description

Get the ADC sampling mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SAMPLING_MODE_NORMAL
  - LL_ADC_REG_SAMPLING_MODE_BULB
  - LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

Reference Manual to LL API cross reference:

- CFGR2 BULB LL_ADC_REG_GetSamplingMode
- CFGR2 SMPTRIG LL_ADC_REG_GetSamplingMode

LL_ADC_REG_StartSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StartSamplingPhase (ADC_TypeDef * ADCx)

Function description

Start ADC sampling phase for sampling time trigger mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger source
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StartSamplingPhase

LL_ADC_REG_StopSamplingPhase

Function name

__STATIC_INLINE void LL_ADC_REG_StopSamplingPhase (ADC_TypeDef * ADCx)

Function description

Stop ADC sampling phase for sampling time trigger mode and start conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This function is relevant only when LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED has been set using LL_ADC_REG_SetSamplingModeLL_ADC_REG_TRIG_SOFTWARE is used as trigger sourceLL_ADC_REG_StartSamplingPhase has been called to start the sampling phase
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CFGR2 SWTRIG LL_ADC_REG_StopSamplingPhase

LL_ADC_REG_SetSequencerConfigurable

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerConfigurable (ADC_TypeDef * ADCx, uint32_t Configurability)

Function description

Set ADC group regular sequencer configuration flexibility.

Parameters

- **ADCx:** ADC instance
- **Configurability:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_FIXED
  - LL_ADC_REG_SEQ_CONFIGURABLE

Return values

- **None:**

Notes

- On this STM32 series, ADC group regular sequencer both modes "fully configurable" or "not fully configurable" are available: sequencer configured to fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().sequencer configured to not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number. Refer to description of function LL_ADC_REG_SetSequencerChannels().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- On this STM32U5 series, this is applicable on ADC4 only.

Reference Manual to LL API cross reference:

- CFGR CHSELRMOD LL_ADC_REG_SetSequencerConfigurable

LL_ADC_REG_GetSequencerConfigurable

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerConfigurable (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer configuration flexibility.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_FIXED
  - LL_ADC_REG_SEQ_CONFIGURABLE

Notes

- On this STM32 series, ADC group regular sequencer both modes "fully configurable" or "not fully configurable" are available: sequencer configured to fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().sequencer configured to not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number. Refer to description of function LL_ADC_REG_SetSequencerChannels().
- On this STM32U5 series, this is applicable on ADC4 only.

Reference Manual to LL API cross reference:

- CFGR CHSELRMOD LL_ADC_REG_SetSequencerConfigurable

LL_ADC_REG_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  – LL_ADC_REG_SEQ_SCAN_DISABLE
  – LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  – LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS Note: Specific case for ADC4, use literals LL_ADC4_REG_SEQ_SCAN_x

Return values

- **None:**

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function performs configuration of: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_SetSequencerLength

LL_ADC_REG_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerLength (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DISABLE
  - LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS
  - LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS Note: Specific case for ADC4, use literals LL_ADC4_REG_SEQ_SCAN_x

Notes

- Description of ADC group regular sequencer features: For devices with sequencer fully configurable (function "LL_ADC_REG_SetSequencerRanks()" available): sequencer length and each rank affectation to a channel are configurable. This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerRanks()". For devices with sequencer not fully configurable (function "LL_ADC_REG_SetSequencerChannels()" available): sequencer length and each rank affectation to a channel are defined by channel number. This function retrieves: Sequence length: Number of ranks in the scan sequence is defined by number of channels set in the sequence, rank of each channel is fixed by channel HW number. (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from lowest channel number to highest channel number). Sequencer ranks are selected using function "LL_ADC_REG_SetSequencerChannels()".
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- SQR1 L LL_ADC_REG_GetSequencerLength

LL_ADC_REG_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Return values

- **None:**

Notes

- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- It is not possible to enable both ADC auto-injected mode and ADC group regular sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_SetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_SetSequencerDiscont

LL_ADC_REG_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_DISCONT_DISABLE
  - LL_ADC_REG_SEQ_DISCONT_1RANK
  - LL_ADC_REG_SEQ_DISCONT_2RANKS
  - LL_ADC_REG_SEQ_DISCONT_3RANKS
  - LL_ADC_REG_SEQ_DISCONT_4RANKS
  - LL_ADC_REG_SEQ_DISCONT_5RANKS
  - LL_ADC_REG_SEQ_DISCONT_6RANKS
  - LL_ADC_REG_SEQ_DISCONT_7RANKS
  - LL_ADC_REG_SEQ_DISCONT_8RANKS

Reference Manual to LL API cross reference:

- CFGR DISCEN LL_ADC_REG_GetSequencerDiscont
- CFGR DISCNUM LL_ADC_REG_GetSequencerDiscont

LL_ADC_REG_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16 Note: Specific case for ADC4, use literals LL_ADC_REG_RANK_x_ADC4

- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **None:**

Notes

- This function performs configuration of: Channels ordering into each rank of scan sequence: whatever channel can be placed into whatever rank.
- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_SetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_SetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_SetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_SetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_SetSequencerRanks

LL_ADC_REG_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_REG_RANK_1
  - LL_ADC_REG_RANK_2
  - LL_ADC_REG_RANK_3
  - LL_ADC_REG_RANK_4
  - LL_ADC_REG_RANK_5
  - LL_ADC_REG_RANK_6
  - LL_ADC_REG_RANK_7
  - LL_ADC_REG_RANK_8
  - LL_ADC_REG_RANK_9
  - LL_ADC_REG_RANK_10
  - LL_ADC_REG_RANK_11
  - LL_ADC_REG_RANK_12
  - LL_ADC_REG_RANK_13
  - LL_ADC_REG_RANK_14
  - LL_ADC_REG_RANK_15
  - LL_ADC_REG_RANK_16 Note: Specific case for ADC4, use literals LL_ADC_REG_RANK_x_ADC4

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_CHANNEL_0 (3)
  – LL_ADC_CHANNEL_1 (3)
  – LL_ADC_CHANNEL_2 (3)
  – LL_ADC_CHANNEL_3 (3)
  – LL_ADC_CHANNEL_4 (3)
  – LL_ADC_CHANNEL_5 (3)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_19
  – LL_ADC_CHANNEL_VREFINT
  – LL_ADC_CHANNEL_TEMPSENSOR (1)
  – LL_ADC_CHANNEL_VBAT (1)
  – LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  – LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  – LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  – LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual). (1, 2) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- On this STM32 series, ADC group regular sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- SQR1 SQ1 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ2 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ3 LL_ADC_REG_GetSequencerRanks
- SQR1 SQ4 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ5 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ6 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ7 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ8 LL_ADC_REG_GetSequencerRanks
- SQR2 SQ9 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ10 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ11 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ12 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ13 LL_ADC_REG_GetSequencerRanks
- SQR3 SQ14 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ15 LL_ADC_REG_GetSequencerRanks
- SQR4 SQ16 LL_ADC_REG_GetSequencerRanks

LL_ADC_REG_SetSequencerScanDirection

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerScanDirection (ADC_TypeDef * ADCx, uint32_t ScanDirection)

Function description

Set ADC group regular sequencer scan direction.

Parameters

- **ADCx:** ADC instance
- **ScanDirection:** This parameter can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DIR_FORWARD
  - LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD

Return values

- **None:**

Notes

- On this STM32 series, parameter relevant only is sequencer is set to mode not fully configurable, refer to function LL_ADC_REG_SetSequencerConfigurable().
- On some other STM32 families, this setting is not available and the default scan direction is forward.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 SCANDIR LL_ADC_REG_SetSequencerScanDirection

LL_ADC_REG_GetSequencerScanDirection

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerScanDirection (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequencer scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_SEQ_SCAN_DIR_FORWARD
  - LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD

Notes

- On this STM32 series, parameter relevant only is sequencer is set to mode not fully configurable, refer to function LL_ADC_REG_SetSequencerConfigurable().
- On some other STM32 families, this setting is not available and the default scan direction is forward.

Reference Manual to LL API cross reference:

- CFGR1 SCANDIR LL_ADC_REG_GetSequencerScanDirection

LL_ADC_REG_SetSequencerChannels

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerChannels (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Set ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15 (1)
  - LL_ADC_CHANNEL_16 (1)
  - LL_ADC_CHANNEL_17 (1)
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VBAT

  (1) On STM32U5, parameter can be set in ADC group sequencer only if sequencer is set in mode "not fully configurable", refer to function LL_ADC_REG_SetSequencerConfigurable().

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Set channels selected by overwriting the current sequencer configuration.
- On this STM32 series, ADC group regular sequencer both modes "fully configurable" or "not fully configurable" are available, they can be chosen using function LL_ADC_REG_SetSequencerConfigurable(). This function can be used with setting "not fully configurable". Refer to description of functions LL_ADC_REG_SetSequencerConfigurable() and LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSELR CHSEL0 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL1 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL2 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL3 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL4 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL5 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL6 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL7 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL8 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL9 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL10 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL11 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL12 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL13 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL14 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL15 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL16 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL17 LL_ADC_REG_SetSequencerChannels
- CHSELR CHSEL18 LL_ADC_REG_SetSequencerChannels

LL_ADC_REG_SetSequencerChAdd

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerChAdd (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Add channel to ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15 (1)
  - LL_ADC_CHANNEL_16 (1)
  - LL_ADC_CHANNEL_17 (1)
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VBAT

  (1) On STM32U5, parameter can be set in ADC group sequencer only if sequencer is set in mode "not fully configurable", refer to function LL_ADC_REG_SetSequencerConfigurable().

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Set channels selected by adding them to the current sequencer configuration.
- On this STM32 series, ADC group regular sequencer both modes "fully configurable" or "not fully configurable" are available, they can be chosen using function LL_ADC_REG_SetSequencerConfigurable(). This function can be used with setting "not fully configurable". Refer to description of functions LL_ADC_REG_SetSequencerConfigurable() and LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSELR CHSEL0 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL1 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL2 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL3 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL4 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL5 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL6 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL7 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL8 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL9 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL10 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL11 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL12 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL13 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL14 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL15 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL16 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL17 LL_ADC_REG_SetSequencerChAdd
- CHSELR CHSEL18 LL_ADC_REG_SetSequencerChAdd

LL_ADC_REG_SetSequencerChRem

Function name

__STATIC_INLINE void LL_ADC_REG_SetSequencerChRem (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Remove channel to ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15 (1)
  - LL_ADC_CHANNEL_16 (1)
  - LL_ADC_CHANNEL_17 (1)
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VBAT

  (1) On STM32U5, parameter can be set in ADC group sequencer only if sequencer is set in mode "not fully configurable", refer to function LL_ADC_REG_SetSequencerConfigurable().

Return values

- **None:**

Notes

- This function performs: Channels ordering into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).Set channels selected by removing them to the current sequencer configuration.
- On this STM32 series, ADC group regular sequencer both modes "fully configurable" or "not fully configurable" are available, they can be chosen using function LL_ADC_REG_SetSequencerConfigurable(). This function can be used with setting "not fully configurable". Refer to description of functions LL_ADC_REG_SetSequencerConfigurable() and LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSELR CHSEL0 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL1 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL2 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL3 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL4 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL5 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL6 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL7 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL8 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL9 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL10 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL11 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL12 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL13 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL14 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL15 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL16 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL17 LL_ADC_REG_SetSequencerChRem
- CHSELR CHSEL18 LL_ADC_REG_SetSequencerChRem

LL_ADC_REG_GetSequencerChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetSequencerChannels (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular sequence: channel on rank corresponding to channel number.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be a combination of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15 (1)
  - LL_ADC_CHANNEL_16 (1)
  - LL_ADC_CHANNEL_17 (1)
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR
  - LL_ADC_CHANNEL_VBAT

  (1) On STM32U5, parameter can be set in ADC group sequencer only if sequencer is set in mode "not fully configurable", refer to function LL_ADC_REG_SetSequencerConfigurable().

Notes

- This function performs: Channels order reading into each rank of scan sequence: rank of each channel is fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...).
- On this STM32 series, ADC group regular sequencer both modes "fully configurable" or "not fully configurable" are available, they can be chosen using function LL_ADC_REG_SetSequencerConfigurable(). This function can be used with setting "not fully configurable". Refer to description of functions LL_ADC_REG_SetSequencerConfigurable() and LL_ADC_REG_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- One or several values can be retrieved. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- CHSELR CHSEL0 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL1 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL2 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL3 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL4 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL5 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL6 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL7 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL8 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL9 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL10 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL11 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL12 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL13 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL14 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL15 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL16 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL17 LL_ADC_REG_GetSequencerChannels
- CHSELR CHSEL18 LL_ADC_REG_GetSequencerChannels

LL_ADC_SetChannelPreselection

Function name

__STATIC_INLINE void LL_ADC_SetChannelPreselection (ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Set ADC Channel Preselection to LL_ADC_CHANNEL_x, x = 0 to 19.

Parameters

- **ADCx:** ADC instance.
- **Channel:** This parameter can be one of the following values:
  – LL_ADC_CHANNEL_0
  – LL_ADC_CHANNEL_1
  – LL_ADC_CHANNEL_2
  – LL_ADC_CHANNEL_3
  – LL_ADC_CHANNEL_4
  – LL_ADC_CHANNEL_5
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_19
  – LL_ADC_CHANNEL_VREFINT
  – LL_ADC_CHANNEL_TEMPSENSOR
  – LL_ADC_CHANNEL_VBAT

Return values

- **None:**

Notes

- This function set the the value for the channel preselection register corresponding to ADC channel to be selected.
- Caution: This is not valid for ADC4.

Reference Manual to LL API cross reference:

- PCSEL PCSEL0 LL_ADC_CHANNEL_0
- PCSEL PCSEL1 LL_ADC_CHANNEL_1
- PCSEL PCSEL2 LL_ADC_CHANNEL_2
- PCSEL PCSEL3 LL_ADC_CHANNEL_3
- PCSEL PCSEL4 LL_ADC_CHANNEL_4
- PCSEL PCSEL5 LL_ADC_CHANNEL_5
- PCSEL PCSEL6 LL_ADC_CHANNEL_6
- PCSEL PCSEL7 LL_ADC_CHANNEL_7
- PCSEL PCSEL8 LL_ADC_CHANNEL_8
- PCSEL PCSEL9 LL_ADC_CHANNEL_9
- PCSEL PCSEL10 LL_ADC_CHANNEL_10
- PCSEL PCSEL11 LL_ADC_CHANNEL_11
- PCSEL PCSEL12 LL_ADC_CHANNEL_12
- PCSEL PCSEL13 LL_ADC_CHANNEL_13
- PCSEL PCSEL14 LL_ADC_CHANNEL_14
- PCSEL PCSEL15 LL_ADC_CHANNEL_15
- PCSEL PCSEL16 LL_ADC_CHANNEL_16
- PCSEL PCSEL17 LL_ADC_CHANNEL_17
- PCSEL PCSEL18 LL_ADC_CHANNEL_18
- PCSEL PCSEL19 LL_ADC_CHANNEL_19

LL_ADC_GetChannelPreselection

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelPreselection (const ADC_TypeDef * ADCx)

Function description

Get ADC Channel Preselection register value.

Parameters

- **ADCx:** ADC instance.

Return values

- **Returned:** decimal value that can correspend to one or multiple channels:

Notes

- This function set the the value for the channel preselection register corresponding to ADC channel to be selected.
- Caution: This is not valid for ADC4.
- User helper macro __LL_ADC_DECIMAL_NB_TO_CHANNEL().

Reference Manual to LL API cross reference:

- PCSEL PCSEL0 LL_ADC_CHANNEL_0
- PCSEL PCSEL1 LL_ADC_CHANNEL_1
- PCSEL PCSEL2 LL_ADC_CHANNEL_2
- PCSEL PCSEL3 LL_ADC_CHANNEL_3
- PCSEL PCSEL4 LL_ADC_CHANNEL_4
- PCSEL PCSEL5 LL_ADC_CHANNEL_5
- PCSEL PCSEL6 LL_ADC_CHANNEL_6
- PCSEL PCSEL7 LL_ADC_CHANNEL_7
- PCSEL PCSEL8 LL_ADC_CHANNEL_8
- PCSEL PCSEL9 LL_ADC_CHANNEL_9
- PCSEL PCSEL10 LL_ADC_CHANNEL_10
- PCSEL PCSEL11 LL_ADC_CHANNEL_11
- PCSEL PCSEL12 LL_ADC_CHANNEL_12
- PCSEL PCSEL13 LL_ADC_CHANNEL_13
- PCSEL PCSEL14 LL_ADC_CHANNEL_14
- PCSEL PCSEL15 LL_ADC_CHANNEL_15
- PCSEL PCSEL16 LL_ADC_CHANNEL_16
- PCSEL PCSEL17 LL_ADC_CHANNEL_17
- PCSEL PCSEL18 LL_ADC_CHANNEL_18
- PCSEL PCSEL19 LL_ADC_CHANNEL_19

LL_ADC_SetLPModeAutoPowerOff

Function name

__STATIC_INLINE void LL_ADC_SetLPModeAutoPowerOff (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  – LL_ADC_LP_AUTOPOWEROFF_DISABLE
  – LL_ADC_LP_AUTOPOWEROFF_ENABLE

Return values

- **None:**

Notes

- Description of ADC low power mode: ADC low power mode "auto power-off": the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled.

Reference Manual to LL API cross reference:

- PWRR AUTOFF LL_ADC_SetLPModeAutoPowerOff

LL_ADC_GetLPModeAutoPowerOff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLPModeAutoPowerOff (const ADC_TypeDef * ADCx)

Function description

Get ADC low power mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_LP_AUTOPOWEROFF_DISABLE
  – LL_ADC_LP_AUTOPOWEROFF_ENABLE

Notes

- Description of ADC low power mode: ADC low power mode "auto power-off": the ADC automatically powers-off after a conversion and automatically wakes up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with low power mode "auto wait".

Reference Manual to LL API cross reference:

- PW AUTOFF LL_ADC_GetLPModeAutoPowerOff

LL_ADC_SetLPModeAutonomousDPD

Function name

__STATIC_INLINE void LL_ADC_SetLPModeAutonomousDPD (ADC_TypeDef * ADCx, uint32_t LowPowerMode)

Function description

Set ADC low power mode: deep power down in autonomous mode.

Parameters

- **ADCx:** ADC instance
- **LowPowerMode:** This parameter can be one of the following values:
  – LL_ADC_LP_AUTONOMOUS_DPD_DISABLE
  – LL_ADC_LP_AUTONOMOUS_DPD_ENABLE

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled.

Reference Manual to LL API cross reference:

- PWRR DPD LL_ADC_SetLPModeAutonomousDPD

LL_ADC_GetLPModeAutonomousDPD

Function name

__STATIC_INLINE uint32_t LL_ADC_GetLPModeAutonomousDPD (const ADC_TypeDef * ADCx)

Function description

Get ADC low power mode: deep power down in autonomous mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_LP_AUTONOMOUS_DPD_DISABLE
  - LL_ADC_LP_AUTONOMOUS_DPD_ENABLE

Reference Manual to LL API cross reference:

- PWRR DPD LL_ADC_GetLPModeAutonomousDPD

LL_ADC_SetVrefProtection

Function name

__STATIC_INLINE void LL_ADC_SetVrefProtection (ADC_TypeDef * ADCx, uint32_t VrefProtection)

Function description

Set ADC VREF protection when multiple ADCs are working simultaneously.

Parameters

- **ADCx:** ADC instance
- **VrefProtection:** This parameter can be one of the following values:
  - LL_ADC_VREF_PROT_DISABLE
  - LL_ADC_VREF_PROT_FIRST_SAMP_ENABLE
  - LL_ADC_VREF_PROT_SECOND_SAMP_ENABLE

Return values

- **None:**

Notes

- In case of simultaneous sampling phase of ADC4 and ADC1/2, ADC4 is put on hold during one or two ADC4 clock cycles to avoid noise on Vref+. ADC state: ADC must be disabled.

Reference Manual to LL API cross reference:

- PWRR VREFPROT LL_ADC_SetVrefProtection
- PWRR VREFSECSMP LL_ADC_SetVrefProtection

LL_ADC_GetVrefProtection

Function name

__STATIC_INLINE uint32_t LL_ADC_GetVrefProtection (const ADC_TypeDef * ADCx)

Function description

ADC VREF protection when multiple ADCs are working simultaneously.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_VREF_PROT_DISABLE
  - LL_ADC_VREF_PROT_FIRST_SAMP_ENABLE
  - LL_ADC_VREF_PROT_SECOND_SAMP_ENABLE

Reference Manual to LL API cross reference:

- PWRR VREFPROT LL_ADC_GetVrefProtection
- PWRR VREFSECSMP LL_ADC_GetVrefProtection

LL_ADC_REG_SetContinuousMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetContinuousMode (ADC_TypeDef * ADCx, uint32_t Continuous)

Function description

Set ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance
- **Continuous:** This parameter can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Return values

- **None:**

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.
- It is not possible to enable both ADC group regular continuous mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_SetContinuousMode

LL_ADC_REG_GetContinuousMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetContinuousMode (const ADC_TypeDef * ADCx)

Function description

Get ADC continuous conversion mode on ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_CONV_SINGLE
  - LL_ADC_REG_CONV_CONTINUOUS

Notes

- Description of ADC continuous conversion mode: single mode: one conversion per triggercontinuous mode: after the first trigger, following conversions launched successively automatically.

Reference Manual to LL API cross reference:

- CFGR CONT LL_ADC_REG_GetContinuousMode

LL_ADC_REG_SetDataTransferMode

Function name

__STATIC_INLINE void LL_ADC_REG_SetDataTransferMode (ADC_TypeDef * ADCx, uint32_t DataTransferMode)

Function description

Set ADC data transfer mode.

Parameters

- **ADCx:** ADC instance
- **DataTransferMode:** This parameter can be one of the following values:
  - LL_ADC_REG_DR_TRANSFER
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED
  - LL_ADC_REG_MDF_TRANSFER

Return values

- **None:**

Notes

- Conversion data can be either: Available in Data RegisterTransferred by DMA in one shot modeTransferred by DMA in circular modeTransferred to MDF data register

Reference Manual to LL API cross reference:

- CFGR DMNGT LL_ADC_REG_SetDataTransferMode

LL_ADC_REG_GetDataTransferMode

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetDataTransferMode (const ADC_TypeDef * ADCx)

Function description

Get ADC data transfer mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_REG_DR_TRANSFER
  - LL_ADC_REG_DMA_TRANSFER_LIMITED
  - LL_ADC_REG_DMA_TRANSFER_UNLIMITED
  - LL_ADC_REG_MDF_TRANSFER

Notes

- Conversion data can be either: Available in Data RegisterTransferred by DMA in one shot modeTransferred by DMA in circular modeTransferred to DFSDM data register

Reference Manual to LL API cross reference:

- CFGR DMNGT LL_ADC_REG_GetDataTransferMode

LL_ADC_REG_SetDMATransfer

Function name

__STATIC_INLINE void LL_ADC_REG_SetDMATransfer (ADC_TypeDef * ADCx, uint32_t DMATransfer)

Function description

Set ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters

- **ADCx:** ADC instance
- **DMATransfer:** This parameter can be one of the following values:
  – LL_ADC_REG_DMA_TRANSFER_NONE
  – LL_ADC_REG_DMA_TRANSFER_LIMITED
  – LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Return values

- **None:**

Notes

- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR1 DMAEN LL_ADC_REG_SetDMATransfer
- CFGR1 DMACFG LL_ADC_REG_SetDMATransfer

LL_ADC_REG_GetDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetDMATransfer (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data transfer: no transfer or transfer by DMA, and DMA requests mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_REG_DMA_TRANSFER_NONE
  – LL_ADC_REG_DMA_TRANSFER_LIMITED
  – LL_ADC_REG_DMA_TRANSFER_UNLIMITED

Notes

- If transfer by DMA selected, specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- To configure DMA source address (peripheral address), use function LL_ADC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CFGR1 DMAEN LL_ADC_REG_GetDMATransfer
- CFGR1 DMACFG LL_ADC_REG_GetDMATransfer

LL_ADC_REG_SetOverrun

Function name

__STATIC_INLINE void LL_ADC_REG_SetOverrun (ADC_TypeDef * ADCx, uint32_t Overrun)

Function description

Set ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance
- **Overrun:** This parameter can be one of the following values:
  – LL_ADC_REG_OVR_DATA_PRESERVED
  – LL_ADC_REG_OVR_DATA_OVERWRITTEN

Return values

- **None:**

Notes

- Compatibility with devices without feature overrun: other devices without this feature have a behavior equivalent to data overwritten. The default setting of overrun is data preserved. Therefore, for compatibility with all devices, parameter overrun should be set to data overwritten.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_SetOverrun

LL_ADC_REG_GetOverrun

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_GetOverrun (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular behavior in case of overrun: data preserved or overwritten.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_REG_OVR_DATA_PRESERVED
  – LL_ADC_REG_OVR_DATA_OVERWRITTEN

Reference Manual to LL API cross reference:

- CFGR OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_INJ_SetTriggerSource

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerSource (ADC_TypeDef * ADCx, uint32_t TriggerSource)

Function description

Set ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  – LL_ADC_INJ_TRIG_SOFTWARE
  – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  – LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  – LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  – LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  – LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  – LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  – LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  – LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2
  – LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2
  – LL_ADC_INJ_TRIG_EXT_LPTIM3_CH1

Return values

- **None:**

Notes

- On this STM32 series, setting trigger source to external trigger also set trigger polarity to rising edge (default setting for compatibility with some ADC on other STM32 families having this setting set by HW default value). In case of need to modify trigger edge, use function LL_ADC_INJ_SetTriggerEdge().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_SetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_SetTriggerSource

LL_ADC_INJ_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerSource (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source: internal (SW start) or from external peripheral (timer event, external interrupt line).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_SOFTWARE
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  - LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  - LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  - LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  - LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  - LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  - LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  - LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2
  - LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2
  - LL_ADC_INJ_TRIG_EXT_LPTIM3_CH1

Notes

- To determine whether group injected trigger source is internal (SW start) or external, without detail of which peripheral is selected as external trigger, (equivalent to "if(LL_ADC_INJ_GetTriggerSource(ADC1) == LL_ADC_INJ_TRIG_SOFTWARE)") use function LL_ADC_INJ_IsTriggerSourceSWStart.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_GetTriggerSource
- JSQR JEXTEN LL_ADC_INJ_GetTriggerSource

LL_ADC_INJ_IsTriggerSourceSWStart

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsTriggerSourceSWStart (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger source internal (SW start) or external.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** "0" if trigger source external trigger Value "1" if trigger source SW start.

Notes

- In case of group injected trigger source set to external trigger, to determine which peripheral is selected as external trigger, use function LL_ADC_INJ_GetTriggerSource.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_IsTriggerSourceSWStart

LL_ADC_INJ_SetTriggerEdge

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTriggerEdge (ADC_TypeDef * ADCx, uint32_t ExternalTriggerEdge)

Function description

Set ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_SetTriggerEdge

LL_ADC_INJ_GetTriggerEdge

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTriggerEdge (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger polarity.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_EXT_RISING
  - LL_ADC_INJ_TRIG_EXT_FALLING
  - LL_ADC_INJ_TRIG_EXT_RISINGFALLING

Reference Manual to LL API cross reference:

- JSQR JEXTEN LL_ADC_INJ_GetTriggerEdge

LL_ADC_INJ_SetSequencerLength

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerLength (ADC_TypeDef * ADCx, uint32_t SequencerNbRanks)

Function description

Set ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance
- **SequencerNbRanks:** This parameter can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Return values

- **None:**

Notes

- This function performs configuration of: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_SetSequencerLength

LL_ADC_INJ_GetSequencerLength

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerLength (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer length and scan direction.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_SEQ_SCAN_DISABLE
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  - LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

Notes

- This function retrieves: Sequence length: Number of ranks in the scan sequence.Sequence direction: Unless specified in parameters, sequencer scan direction is forward (from rank 1 to rank n).
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- JSQR JL LL_ADC_INJ_GetSequencerLength

LL_ADC_INJ_SetSequencerDiscont

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerDiscont (ADC_TypeDef * ADCx, uint32_t SeqDiscont)

Function description

Set ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance
- **SeqDiscont:** This parameter can be one of the following values:
  – LL_ADC_INJ_SEQ_DISCONT_DISABLE
  – LL_ADC_INJ_SEQ_DISCONT_1RANK

Return values

- **None:**

Notes

- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_SetSequencerDiscont

LL_ADC_INJ_GetSequencerDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected sequencer discontinuous mode: sequence subdivided and scan conversions interrupted every selected number of ranks.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_INJ_SEQ_DISCONT_DISABLE
  – LL_ADC_INJ_SEQ_DISCONT_1RANK

Reference Manual to LL API cross reference:

- CFGR JDISCEN LL_ADC_INJ_GetSequencerDiscont

LL_ADC_INJ_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_INJ_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **None:**

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32U5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_SetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_SetSequencerRanks

LL_ADC_INJ_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected sequence: channel on the selected sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual). (1, 2) For ADC channel read back from ADC register, comparison with internal channel parameter to be done using helper macro __LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL().

Notes

- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.
- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB().

Reference Manual to LL API cross reference:

- JSQR JSQ1 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ2 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ3 LL_ADC_INJ_GetSequencerRanks
- JSQR JSQ4 LL_ADC_INJ_GetSequencerRanks

LL_ADC_INJ_SetTrigAuto

Function name

__STATIC_INLINE void LL_ADC_INJ_SetTrigAuto (ADC_TypeDef * ADCx, uint32_t TrigAuto)

Function description

Set ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance
- **TrigAuto:** This parameter can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Return values

- **None:**

Notes

- This mode can be used to extend number of data registers updated after one ADC conversion trigger and with data permanently kept (not erased by successive conversions of scan of ADC sequencer ranks), up to 5 data registers: 1 data register on ADC group regular, 4 data registers on ADC group injected.
- If ADC group injected injected trigger source is set to an external trigger, this feature must be must be set to independent trigger. ADC group injected automatic trigger is compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.
- It is not possible to enable both ADC group injected auto-injected mode and sequencer discontinuous mode.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_SetTrigAuto

LL_ADC_INJ_GetTrigAuto

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_GetTrigAuto (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion trigger: independent or from ADC group regular.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_INJ_TRIG_INDEPENDENT
  - LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

Reference Manual to LL API cross reference:

- CFGR JAUTO LL_ADC_INJ_GetTrigAuto

LL_ADC_INJ_ConfigQueueContext

Function name

__STATIC_INLINE void LL_ADC_INJ_ConfigQueueContext (ADC_TypeDef * ADCx, uint32_t TriggerSource, uint32_t ExternalTriggerEdge, uint32_t SequencerNbRanks, uint32_t Rank1_Channel, uint32_t Rank2_Channel, uint32_t Rank3_Channel, uint32_t Rank4_Channel)

Function description

Set one context on ADC group injected that will be checked in contexts queue.

Parameters

- **ADCx:** ADC instance
- **TriggerSource:** This parameter can be one of the following values:
  – LL_ADC_INJ_TRIG_SOFTWARE
  – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM1_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM2_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM2_CH1
  – LL_ADC_INJ_TRIG_EXT_TIM3_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM4_TRGO
  – LL_ADC_INJ_TRIG_EXT_EXTI_LINE15
  – LL_ADC_INJ_TRIG_EXT_TIM8_CH4
  – LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2
  – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2
  – LL_ADC_INJ_TRIG_EXT_TIM3_CH3
  – LL_ADC_INJ_TRIG_EXT_TIM3_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM3_CH1
  – LL_ADC_INJ_TRIG_EXT_TIM6_TRGO
  – LL_ADC_INJ_TRIG_EXT_TIM15_TRGO
  – LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2
  – LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2
  – LL_ADC_INJ_TRIG_EXT_LPTIM3_CH1
- **ExternalTriggerEdge:** This parameter can be one of the following values:
  – LL_ADC_INJ_TRIG_EXT_RISING
  – LL_ADC_INJ_TRIG_EXT_FALLING
  – LL_ADC_INJ_TRIG_EXT_RISINGFALLING

  Note: This parameter is discarded in case of SW start: parameter "TriggerSource" set to "LL_ADC_INJ_TRIG_SOFTWARE".
- **SequencerNbRanks:** This parameter can be one of the following values:
  – LL_ADC_INJ_SEQ_SCAN_DISABLE
  – LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS
  – LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS
  – LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

- **Rank1_Channel:** This parameter can be one of the following values:
  – LL_ADC_CHANNEL_0 (3)
  – LL_ADC_CHANNEL_1 (3)
  – LL_ADC_CHANNEL_2 (3)
  – LL_ADC_CHANNEL_3 (3)
  – LL_ADC_CHANNEL_4 (3)
  – LL_ADC_CHANNEL_5 (3)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_19
  – LL_ADC_CHANNEL_VREFINT
  – LL_ADC_CHANNEL_TEMPSENSOR (1)
  – LL_ADC_CHANNEL_VBAT (1)
  – LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  – LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

- **Rank2_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

- **Rank3_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

- **Rank4_Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **None:**

Notes

- A context is a setting of group injected sequencer: group injected triggersequencer lengthsequencer ranks This function is intended to be used when contexts queue is enabled, because the sequence must be fully configured in one time (functions to set separately injected trigger and sequencer channels cannot be used):
- In the contexts queue, only the active context can be read. The parameters of this function can be read using functions: LL_ADC_INJ_GetTriggerSource() LL_ADC_INJ_GetTriggerEdge() LL_ADC_INJ_GetSequencerRanks()
- On this STM32 series, to measure internal channels (VrefInt, TempSensor, ...), measurement paths to internal channels must be enabled separately. This can be done using function LL_ADC_SetCommonPathInternalCh().
- On STM32U5, some fast channels are available: fast analog inputs coming from GPIO pads (ADC_IN0..5).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must not be disabled. Can be enabled with or without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- JSQR JEXTSEL LL_ADC_INJ_ConfigQueueContext
- JSQR JEXTEN LL_ADC_INJ_ConfigQueueContext
- JSQR JL LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ1 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ2 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ3 LL_ADC_INJ_ConfigQueueContext
- JSQR JSQ4 LL_ADC_INJ_ConfigQueueContext

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTime)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  – LL_ADC_CHANNEL_0 (3)
  – LL_ADC_CHANNEL_1 (3)
  – LL_ADC_CHANNEL_2 (3)
  – LL_ADC_CHANNEL_3 (3)
  – LL_ADC_CHANNEL_4 (3)
  – LL_ADC_CHANNEL_5 (3)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_19
  – LL_ADC_CHANNEL_VREFINT
  – LL_ADC_CHANNEL_TEMPSENSOR (1)
  – LL_ADC_CHANNEL_VBAT (1)
  – LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  – LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  – LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  – LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).
- **SamplingTime:** This parameter can be one of the following values, In case of ADC1 instance:
  – LL_ADC_SAMPLINGTIME_5CYCLES (1)
  – LL_ADC_SAMPLINGTIME_6CYCLES (1)
  – LL_ADC_SAMPLINGTIME_12CYCLES (1)
  – LL_ADC_SAMPLINGTIME_20CYCLES (1)
  – LL_ADC_SAMPLINGTIME_36CYCLES (1)
  – LL_ADC_SAMPLINGTIME_68CYCLES (1)
  – LL_ADC_SAMPLINGTIME_391CYCLES (1)
  – LL_ADC_SAMPLINGTIME_814CYCLES (1)
  – LL_ADC_SAMPLINGTIME_COMMON_1 (2)
  – LL_ADC_SAMPLINGTIME_COMMON_2 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4.

Return values

- **None:**

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- In case of internal channel (VrefInt, TempSensor, ...) to be converted: sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values (parameters TS_vrefint, TS_temp, ...).
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits
- In case of ADC conversion of internal channel (VrefInt, temperature sensor, ...), a sampling time minimum value is required. Refer to device datasheet.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_SetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_SetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_SetChannelSamplingTime

LL_ADC_GetChannelSamplingTime

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSamplingTime (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4. (3) On STM32U5, fast channel (0.125 us for 14-bit resolution (ADC conversion rate up to 8 Ms/s)). Other channels are slow channels (conversion rate: refer to reference manual).

Return values

- **In:** case of ADC1 insatnace, Returned value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_5CYCLES (1)
  - LL_ADC_SAMPLINGTIME_6CYCLES (1)
  - LL_ADC_SAMPLINGTIME_12CYCLES (1)
  - LL_ADC_SAMPLINGTIME_20CYCLES (1)
  - LL_ADC_SAMPLINGTIME_36CYCLES (1)
  - LL_ADC_SAMPLINGTIME_68CYCLES (1)
  - LL_ADC_SAMPLINGTIME_391CYCLES (1)
  - LL_ADC_SAMPLINGTIME_814CYCLES (1)
  - LL_ADC_SAMPLINGTIME_COMMON_1 (2)
  - LL_ADC_SAMPLINGTIME_COMMON_2 (2)

  (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4.

Notes

- On this device, sampling time is on channel scope: independently of channel mapped on ADC group regular or injected.
- Conversion time is the addition of sampling time and processing time. On this STM32 series, ADC processing time is: 12.5 ADC clock cycles at ADC resolution 12 bits10.5 ADC clock cycles at ADC resolution 10 bits8.5 ADC clock cycles at ADC resolution 8 bits6.5 ADC clock cycles at ADC resolution 6 bits

Reference Manual to LL API cross reference:

- SMPR1 SMP0 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP1 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP2 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP3 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP4 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP5 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP6 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP7 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP8 LL_ADC_GetChannelSamplingTime
- SMPR1 SMP9 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP10 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP11 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP12 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP13 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP14 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP15 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP16 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP17 LL_ADC_GetChannelSamplingTime
- SMPR2 SMP18 LL_ADC_GetChannelSamplingTime

LL_ADC_SetChannelSingleDiff

Function name

__STATIC_INLINE void LL_ADC_SetChannelSingleDiff (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SingleDiff)

Function description

Set mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
- **SingleDiff:** This parameter can be a combination of the following values:
  - LL_ADC_SINGLE_ENDED
  - LL_ADC_DIFFERENTIAL_ENDED

Return values

- **None:**

Notes

- Channel ending is on channel scope: independently of channel mapped on ADC group regular or injected. In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input). Only channel 'i' has to be configured, channel 'i+1' is configured automatically.
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32U5, some channels are internally fixed to single-ended inputs configuration: ADC1: Channels 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17ADC2: Channels 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17ADC4: None
- For ADC channels configured in differential mode, both inputs should be biased at (Vref+)/2 +/-200mV. (Vref+ is the analog voltage reference)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.
- One or several values can be selected. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_SetChannelSingleDiff

LL_ADC_GetChannelSingleDiff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelSingleDiff (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get mode single-ended or differential input of the selected ADC channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination of the following values:
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Return values

- **0:** channel in single-ended mode, else: channel in differential mode

Notes

- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately. Therefore, to ensure a channel is configured in single-ended mode, the configuration of channel itself and the channel 'i-1' must be read back (to ensure that the selected channel channel has not been configured in differential mode by the previous channel).
- Refer to Reference Manual to ensure the selected channel is available in differential mode. For example, internal channels (VrefInt, TempSensor, ...) are not available in differential mode.
- When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
- On STM32U5, some channels are internally fixed to single-ended inputs configuration: ADC1: Channels 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17ADC2: Channels 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and 17ADC4: None
- One or several values can be selected. In this case, the value returned is null if all channels are in single ended-mode. Example: (LL_ADC_CHANNEL_4 | LL_ADC_CHANNEL_12 | ...)

Reference Manual to LL API cross reference:

- DIFSEL DIFSEL LL_ADC_GetChannelSingleDiff

LL_ADC_SetAnalogWDMonitChannels

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDMonitChannels (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDChannelGroup)

Function description

Set ADC analog watchdog monitored channels: a single channel, multiple channels or all channels, on ADC groups regular and-or injected.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2
  - LL_ADC_AWD3

- **AWDChannelGroup:** This parameter can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)
  - LL_ADC_AWD_CHANNEL_14_REG_INJ

- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CHANNEL_19_REG (0)
- LL_ADC_AWD_CHANNEL_19_INJ (0)
- LL_ADC_AWD_CHANNEL_19_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (0)(1)
- LL_ADC_AWD_CH_VREFINT_INJ (0)(1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (0)(1)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (0)(1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (1)
- LL_ADC_AWD_CH_VBAT_REG (0)(1)
- LL_ADC_AWD_CH_VBAT_INJ (0)(1)
- LL_ADC_AWD_CH_VBAT_REG_INJ (1)
- LL_ADC_AWD_CH_DAC1CH1_ADC4_REG (0)(2)
- LL_ADC_AWD_CH_DAC1CH2_ADC4_REG (0)(2)

(0) On STM32U5, parameter available only on analog watchdog number: AWD1. (1) On STM32U5, parameter available only on ADC instance: ADC1, ADC2. (2) On STM32U5, parameter available only on ADC instance: ADC4.

Return values

- **None:**

Notes

- Once monitored channels are selected, analog watchdog is enabled.
- In case of need to define a single channel to monitor with analog watchdog from sequencer channel definition, use helper macro __LL_ADC_ANALOGWD_CHANNEL_GROUP().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_SetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_SetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_SetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_SetAnalogWDMonitChannels

LL_ADC_GetAnalogWDMonitChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDMonitChannels (const ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog monitored channel.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  - LL_ADC_AWD1
  - LL_ADC_AWD2 (1)
  - LL_ADC_AWD3 (1)

  (1) On this AWD number, monitored channel can be retrieved if only 1 channel is programmed (or none or all channels). This function cannot retrieve monitored channel if multiple channels are programmed simultaneously by bitfield.

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG (0)
  - LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  - LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  - LL_ADC_AWD_CHANNEL_0_REG (0)
  - LL_ADC_AWD_CHANNEL_0_INJ (0)
  - LL_ADC_AWD_CHANNEL_0_REG_INJ
  - LL_ADC_AWD_CHANNEL_1_REG (0)
  - LL_ADC_AWD_CHANNEL_1_INJ (0)
  - LL_ADC_AWD_CHANNEL_1_REG_INJ
  - LL_ADC_AWD_CHANNEL_2_REG (0)
  - LL_ADC_AWD_CHANNEL_2_INJ (0)
  - LL_ADC_AWD_CHANNEL_2_REG_INJ
  - LL_ADC_AWD_CHANNEL_3_REG (0)
  - LL_ADC_AWD_CHANNEL_3_INJ (0)
  - LL_ADC_AWD_CHANNEL_3_REG_INJ
  - LL_ADC_AWD_CHANNEL_4_REG (0)
  - LL_ADC_AWD_CHANNEL_4_INJ (0)
  - LL_ADC_AWD_CHANNEL_4_REG_INJ
  - LL_ADC_AWD_CHANNEL_5_REG (0)
  - LL_ADC_AWD_CHANNEL_5_INJ (0)
  - LL_ADC_AWD_CHANNEL_5_REG_INJ
  - LL_ADC_AWD_CHANNEL_6_REG (0)
  - LL_ADC_AWD_CHANNEL_6_INJ (0)
  - LL_ADC_AWD_CHANNEL_6_REG_INJ
  - LL_ADC_AWD_CHANNEL_7_REG (0)
  - LL_ADC_AWD_CHANNEL_7_INJ (0)
  - LL_ADC_AWD_CHANNEL_7_REG_INJ
  - LL_ADC_AWD_CHANNEL_8_REG (0)
  - LL_ADC_AWD_CHANNEL_8_INJ (0)
  - LL_ADC_AWD_CHANNEL_8_REG_INJ
  - LL_ADC_AWD_CHANNEL_9_REG (0)
  - LL_ADC_AWD_CHANNEL_9_INJ (0)
  - LL_ADC_AWD_CHANNEL_9_REG_INJ
  - LL_ADC_AWD_CHANNEL_10_REG (0)
  - LL_ADC_AWD_CHANNEL_10_INJ (0)
  - LL_ADC_AWD_CHANNEL_10_REG_INJ
  - LL_ADC_AWD_CHANNEL_11_REG (0)
  - LL_ADC_AWD_CHANNEL_11_INJ (0)
  - LL_ADC_AWD_CHANNEL_11_REG_INJ
  - LL_ADC_AWD_CHANNEL_12_REG (0)
  - LL_ADC_AWD_CHANNEL_12_INJ (0)
  - LL_ADC_AWD_CHANNEL_12_REG_INJ
  - LL_ADC_AWD_CHANNEL_13_REG (0)
  - LL_ADC_AWD_CHANNEL_13_INJ (0)
  - LL_ADC_AWD_CHANNEL_13_REG_INJ
  - LL_ADC_AWD_CHANNEL_14_REG (0)
  - LL_ADC_AWD_CHANNEL_14_INJ (0)

- – LL_ADC_AWD_CHANNEL_14_REG_INJ
- – LL_ADC_AWD_CHANNEL_15_REG (0)
- – LL_ADC_AWD_CHANNEL_15_INJ (0)
- – LL_ADC_AWD_CHANNEL_15_REG_INJ
- – LL_ADC_AWD_CHANNEL_16_REG (0)
- – LL_ADC_AWD_CHANNEL_16_INJ (0)
- – LL_ADC_AWD_CHANNEL_16_REG_INJ
- – LL_ADC_AWD_CHANNEL_17_REG (0)
- – LL_ADC_AWD_CHANNEL_17_INJ (0)
- – LL_ADC_AWD_CHANNEL_17_REG_INJ
- – LL_ADC_AWD_CHANNEL_18_REG (0)
- – LL_ADC_AWD_CHANNEL_18_INJ (0)
- – LL_ADC_AWD_CHANNEL_18_REG_INJ
- – LL_ADC_AWD_CHANNEL_19_REG (0)
- – LL_ADC_AWD_CHANNEL_19_INJ (0)
- – LL_ADC_AWD_CHANNEL_19_REG_INJ

(0) On STM32U5, parameter available only on analog watchdog number: AWD1.

Notes

- Usage of the returned channel number: To reinject this channel into another function LL_ADC_xxx: the returned channel number is only partly formatted on definition of literals LL_ADC_CHANNEL_x. Therefore, it has to be compared with parts of literals LL_ADC_CHANNEL_x or using helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Then the selected literal LL_ADC_CHANNEL_x can be used as parameter for another function.To get the channel number in decimal format: process the returned value with the helper macro __LL_ADC_CHANNEL_TO_DECIMAL_NB(). Applicable only when the analog watchdog is set to monitor one channel.
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CFGR AWD1EN LL_ADC_GetAnalogWDMonitChannels
- CFGR JAWD1EN LL_ADC_GetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

LL_ADC_SetAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_SetAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow, uint32_t AWDThresholdValue)

Function description

Set ADC analog watchdog threshold value of threshold high or low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  – LL_ADC_AWD1
  – LL_ADC_AWD2
  – LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  – LL_ADC_AWD_THRESHOLD_HIGH
  – LL_ADC_AWD_THRESHOLD_LOW
- **AWDThresholdValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC groups regular and-or injected.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is limited to 8 bits: if ADC resolution is 12 bits the 4 LSB are ignored, if ADC resolution is 10 bits the 2 LSB are ignored.
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling intermediate computation (after ratio, before shift application): intermediate register bitfield [32:7] (26 most significant bits).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either ADC groups regular or injected.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_SetAnalogWDThresholds
- TR2 HT2 LL_ADC_SetAnalogWDThresholds
- TR3 HT3 LL_ADC_SetAnalogWDThresholds
- TR1 LT1 LL_ADC_SetAnalogWDThresholds
- TR2 LT2 LL_ADC_SetAnalogWDThresholds
- TR3 LT3 LL_ADC_SetAnalogWDThresholds

LL_ADC_GetAnalogWDThresholds

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAnalogWDThresholds (const ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdsHighLow)

Function description

Get ADC analog watchdog threshold value of threshold high, threshold low or raw data with ADC thresholds high and low concatenated.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  – LL_ADC_AWD1
  – LL_ADC_AWD2
  – LL_ADC_AWD3
- **AWDThresholdsHighLow:** This parameter can be one of the following values:
  – LL_ADC_AWD_THRESHOLD_HIGH
  – LL_ADC_AWD_THRESHOLD_LOW

Return values

- **In:** case of ADC1 instance, Value between Min_Data=0x000 and Max_Data=0x1FFFFFF
- **In:** case of ADC1 instance, Value between Min_Data=0x000 and Max_Data=0xFFF

Notes

- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION().

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_GetAnalogWDThresholds
- TR2 HT2 LL_ADC_GetAnalogWDThresholds
- TR3 HT3 LL_ADC_GetAnalogWDThresholds
- TR1 LT1 LL_ADC_GetAnalogWDThresholds
- TR2 LT2 LL_ADC_GetAnalogWDThresholds
- TR3 LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_ConfigAnalogWDThresholds

Function name

__STATIC_INLINE void LL_ADC_ConfigAnalogWDThresholds (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t AWDThresholdHighValue, uint32_t AWDThresholdLowValue)

Function description

Set ADC analog watchdog thresholds value of both thresholds high and low.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  – LL_ADC_AWD1
  – LL_ADC_AWD2
  – LL_ADC_AWD3
- **AWDThresholdHighValue:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **AWDThresholdLowValue:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Notes

- If value of only one threshold high or low must be set, use function LL_ADC_SetAnalogWDThresholds().
- In case of ADC resolution different of 12 bits, analog watchdog thresholds data require a specific shift. Use helper macro __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION().
- On this STM32 series, there are 2 kinds of analog watchdog instance: AWD standard (instance AWD1): channels monitored: can monitor 1 channel or all channels.groups monitored: ADC group regular.resolution: resolution is not limited (corresponds to ADC resolution configured). AWD flexible (instances AWD2, AWD3): channels monitored: flexible on channels monitored, selection is channel wise, from from 1 to all channels. Specificity of this analog watchdog: Multiple channels can be selected. For example: (LL_ADC_AWD_CHANNEL4_REG_INJ | LL_ADC_AWD_CHANNEL5_REG_INJ | ...)groups monitored: not selection possible (monitoring on both groups regular and injected). Channels selected are monitored on groups regular and injected: LL_ADC_AWD_CHANNELxx_REG_INJ (do not use parameters LL_ADC_AWD_CHANNELxx_REG and LL_ADC_AWD_CHANNELxx_INJ)resolution: resolution is not limited (corresponds to ADC resolution configured).
- If ADC oversampling is enabled, ADC analog watchdog thresholds are impacted: the comparison of analog watchdog thresholds is done on oversampling final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits). Examples: Oversampling ratio and shift selected to have ADC conversion data on 12 bits (ratio 16 and shift 4, or ratio 32 and shift 5, ...): ADC analog watchdog thresholds must be divided by 16.Oversampling ratio and shift selected to have ADC conversion data on 14 bits (ratio 16 and shift 2, or ratio 32 and shift 3, ...): ADC analog watchdog thresholds must be divided by 4.Oversampling ratio and shift selected to have ADC conversion data on 16 bits (ratio 16 and shift none, or ratio 32 and shift 1, ...): ADC analog watchdog thresholds match directly to ADC data register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- TR1 HT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 HT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 HT3 LL_ADC_ConfigAnalogWDThresholds
- TR1 LT1 LL_ADC_ConfigAnalogWDThresholds
- TR2 LT2 LL_ADC_ConfigAnalogWDThresholds
- TR3 LT3 LL_ADC_ConfigAnalogWDThresholds

LL_ADC_SetAWDFilteringConfiguration

Function name

__STATIC_INLINE void LL_ADC_SetAWDFilteringConfiguration (ADC_TypeDef * ADCx, uint32_t AWDy, uint32_t FilteringConfig)

Function description

Set ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  – LL_ADC_AWD1
- **FilteringConfig:** This parameter can be one of the following values:
  – LL_ADC_AWD_FILTERING_NONE
  – LL_ADC_AWD_FILTERING_2SAMPLES
  – LL_ADC_AWD_FILTERING_3SAMPLES
  – LL_ADC_AWD_FILTERING_4SAMPLES
  – LL_ADC_AWD_FILTERING_5SAMPLES
  – LL_ADC_AWD_FILTERING_6SAMPLES
  – LL_ADC_AWD_FILTERING_7SAMPLES
  – LL_ADC_AWD_FILTERING_8SAMPLES

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.
- On this STM32 series, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- TR1 AWDFILT LL_ADC_SetAWDFilteringConfiguration

LL_ADC_GetAWDFilteringConfiguration

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDFilteringConfiguration (const ADC_TypeDef * ADCx, uint32_t AWDy)

Function description

Get ADC analog watchdog filtering configuration.

Parameters

- **ADCx:** ADC instance
- **AWDy:** This parameter can be one of the following values:
  – LL_ADC_AWD1

Return values

- **Returned:** value can be:
  – LL_ADC_AWD_FILTERING_NONE
  – LL_ADC_AWD_FILTERING_2SAMPLES
  – LL_ADC_AWD_FILTERING_3SAMPLES
  – LL_ADC_AWD_FILTERING_4SAMPLES
  – LL_ADC_AWD_FILTERING_5SAMPLES
  – LL_ADC_AWD_FILTERING_6SAMPLES
  – LL_ADC_AWD_FILTERING_7SAMPLES
  – LL_ADC_AWD_FILTERING_8SAMPLES

Notes

- On this STM32 series, this feature is only available on first analog watchdog (AWD1)

Reference Manual to LL API cross reference:

- TR1 AWDFILT LL_ADC_GetAWDFilteringConfiguration

LL_ADC_SetOverSamplingScope

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)

Function description

Set ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).

Parameters

- **ADCx:** ADC instance
- **OvsScope:** This parameter can be one of the following values:
  – LL_ADC_OVS_DISABLE
  – LL_ADC_OVS_GRP_REGULAR_CONTINUED
  – LL_ADC_OVS_GRP_REGULAR_RESUMED
  – LL_ADC_OVS_GRP_INJECTED
  – LL_ADC_OVS_GRP_INJ_REG_RESUMED

Return values

- **None:**

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_SetOverSamplingScope
- CFGR2 JOVSE LL_ADC_SetOverSamplingScope
- CFGR2 ROVSM LL_ADC_SetOverSamplingScope

LL_ADC_GetOverSamplingScope

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingScope (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling scope: ADC groups regular and-or injected (availability of ADC group injected depends on STM32 families).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_OVS_DISABLE
  – LL_ADC_OVS_GRP_REGULAR_CONTINUED
  – LL_ADC_OVS_GRP_REGULAR_RESUMED
  – LL_ADC_OVS_GRP_INJECTED
  – LL_ADC_OVS_GRP_INJ_REG_RESUMED

Notes

- If both groups regular and injected are selected, specify behavior of ADC group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is either temporary stopped and continued, or resumed from start (oversampler buffer reset).

Reference Manual to LL API cross reference:

- CFGR2 ROVSE LL_ADC_GetOverSamplingScope
- CFGR2 JOVSE LL_ADC_GetOverSamplingScope
- CFGR2 ROVSM LL_ADC_GetOverSamplingScope

LL_ADC_SetOverSamplingDiscont

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingDiscont (ADC_TypeDef * ADCx, uint32_t OverSamplingDiscont)

Function description

Set ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance
- **OverSamplingDiscont:** This parameter can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Return values

- **None:**

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on group regular.
- On this STM32 series, oversampling discontinuous mode (triggered mode) can be used only when oversampling is set on group regular only and in resumed mode.

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_SetOverSamplingDiscont

LL_ADC_GetOverSamplingDiscont

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingDiscont (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling discontinuous mode (triggered mode) on the selected ADC group.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_OVS_REG_CONT
  - LL_ADC_OVS_REG_DISCONT

Notes

- Number of oversampled conversions are done either in: continuous mode (all conversions of oversampling ratio are done from 1 trigger)discontinuous mode (each conversion of oversampling ratio needs a trigger)

Reference Manual to LL API cross reference:

- CFGR2 TROVS LL_ADC_GetOverSamplingDiscont

LL_ADC_ConfigOverSamplingRatioShift

Function name

__STATIC_INLINE void LL_ADC_ConfigOverSamplingRatioShift (ADC_TypeDef * ADCx, uint32_t Ratio, uint32_t Shift)

Function description

Set ADC oversampling (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance
- **Ratio:** For ADC instance ADC1, ADC2: This parameter can be in the range from 1 to 1024. For ADC instance ADC4: This parameter can be one of the following values:
  – LL_ADC_OVS_RATIO_2
  – LL_ADC_OVS_RATIO_4
  – LL_ADC_OVS_RATIO_8
  – LL_ADC_OVS_RATIO_16
  – LL_ADC_OVS_RATIO_32
  – LL_ADC_OVS_RATIO_64
  – LL_ADC_OVS_RATIO_128
  – LL_ADC_OVS_RATIO_256
- **Shift:** This parameter can be one of the following values:
  – LL_ADC_OVS_SHIFT_NONE
  – LL_ADC_OVS_SHIFT_RIGHT_1
  – LL_ADC_OVS_SHIFT_RIGHT_2
  – LL_ADC_OVS_SHIFT_RIGHT_3
  – LL_ADC_OVS_SHIFT_RIGHT_4
  – LL_ADC_OVS_SHIFT_RIGHT_5
  – LL_ADC_OVS_SHIFT_RIGHT_6
  – LL_ADC_OVS_SHIFT_RIGHT_7
  – LL_ADC_OVS_SHIFT_RIGHT_8
  – LL_ADC_OVS_SHIFT_RIGHT_9 (1)
  – LL_ADC_OVS_SHIFT_RIGHT_10 (1)
  – LL_ADC_OVS_SHIFT_RIGHT_11 (1)

  (1): Only for ADC1 instance.

Return values

- **None:**

Notes

- This function set the 2 items of oversampling configuration: ratioshift
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled or enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_ConfigOverSamplingRatioShift
- CFGR2 OVSR LL_ADC_ConfigOverSamplingRatioShift

LL_ADC_GetOverSamplingRatio

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingRatio (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling ratio (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Ratio:** This parameter can be a value from 1 to 1024 in the case of ADC1/2, In the case of ADC4 can be one of the following values:
  - LL_ADC_OVS_RATIO_2
  - LL_ADC_OVS_RATIO_4
  - LL_ADC_OVS_RATIO_8
  - LL_ADC_OVS_RATIO_16
  - LL_ADC_OVS_RATIO_32
  - LL_ADC_OVS_RATIO_64
  - LL_ADC_OVS_RATIO_128
  - LL_ADC_OVS_RATIO_256

Reference Manual to LL API cross reference:

- CFGR2 OVSR LL_ADC_GetOverSamplingRatio

LL_ADC_GetOverSamplingShift

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverSamplingShift (const ADC_TypeDef * ADCx)

Function description

Get ADC oversampling shift (impacting both ADC groups regular and injected)

Parameters

- **ADCx:** ADC instance

Return values

- **Shift:** This parameter can be one of the following values:
  - LL_ADC_OVS_SHIFT_NONE
  - LL_ADC_OVS_SHIFT_RIGHT_1
  - LL_ADC_OVS_SHIFT_RIGHT_2
  - LL_ADC_OVS_SHIFT_RIGHT_3
  - LL_ADC_OVS_SHIFT_RIGHT_4
  - LL_ADC_OVS_SHIFT_RIGHT_5
  - LL_ADC_OVS_SHIFT_RIGHT_6
  - LL_ADC_OVS_SHIFT_RIGHT_7
  - LL_ADC_OVS_SHIFT_RIGHT_8
  - LL_ADC_OVS_SHIFT_RIGHT_9 (1)
  - LL_ADC_OVS_SHIFT_RIGHT_10 (1)
  - LL_ADC_OVS_SHIFT_RIGHT_11 (1)

  (1): Only for ADC1 instance.

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_GetOverSamplingShift

LL_ADC_SetMultimode

Function name

__STATIC_INLINE void LL_ADC_SetMultimode (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t Multimode)

Function description

Set ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **Multimode:** This parameter can be one of the following values:
  – LL_ADC_MULTI_INDEPENDENT
  – LL_ADC_MULTI_DUAL_REG_SIMULT
  – LL_ADC_MULTI_DUAL_REG_INTERL
  – LL_ADC_MULTI_DUAL_INJ_SIMULT
  – LL_ADC_MULTI_DUAL_INJ_ALTERN
  – LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  – LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  – LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Return values

- **None:**

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_SetMultimode

LL_ADC_GetMultimode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultimode (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode configuration to operate in independent mode or multimode (for devices with several ADC instances).

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_MULTI_INDEPENDENT
  – LL_ADC_MULTI_DUAL_REG_SIMULT
  – LL_ADC_MULTI_DUAL_REG_INTERL
  – LL_ADC_MULTI_DUAL_INJ_SIMULT
  – LL_ADC_MULTI_DUAL_INJ_ALTERN
  – LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM
  – LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT
  – LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

Notes

- If multimode configuration: the selected ADC instance is either master or slave depending on hardware. Refer to reference manual.

Reference Manual to LL API cross reference:

- CCR DUAL LL_ADC_GetMultimode

LL_ADC_SetMultiDMATransfer

Function name

__STATIC_INLINE void LL_ADC_SetMultiDMATransfer (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiDMATransfer)

Function description

Set ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiDMATransfer:** This parameter can be one of the following values:
  – LL_ADC_MULTI_REG_DMA_EACH_ADC
  – LL_ADC_MULTI_REG_DMA_RES_32_10B
  – LL_ADC_MULTI_REG_DMA_RES_8B

Return values

- **None:**

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled or enabled without conversion on going on group regular.

Reference Manual to LL API cross reference:

- CCR DAMDF LL_ADC_GetMultiDMATransfer
-

LL_ADC_GetMultiDMATransfer

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiDMATransfer (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode conversion data transfer: no transfer or transfer by DMA.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_REG_DMA_EACH_ADC
  - LL_ADC_MULTI_REG_DMA_RES_32_10B
  - LL_ADC_MULTI_REG_DMA_RES_8B

Notes

- If ADC multimode transfer by DMA is not selected: each ADC uses its own DMA channel, with its individual DMA transfer settings. If ADC multimode transfer by DMA is selected: One DMA channel is used for both ADC (DMA of ADC master) Specifies the DMA requests mode: Limited mode (One shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.Unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transfers (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.
- If ADC DMA requests mode is set to unlimited and DMA is set to mode non-circular: when DMA transfers size will be reached, DMA will stop transfers of ADC conversions data ADC will raise an overrun error (overrun flag and interruption if enabled).
- How to retrieve multimode conversion data: Whatever multimode transfer by DMA setting: using function LL_ADC_REG_ReadMultiConversionData32(). If ADC multimode transfer by DMA is selected: conversion data is a raw data with ADC master and slave concatenated. A macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE().

Reference Manual to LL API cross reference:

- CCR DAMDF LL_ADC_GetMultiDMATransfer
-

LL_ADC_SetMultiTwoSamplingDelay

Function name

__STATIC_INLINE void LL_ADC_SetMultiTwoSamplingDelay (ADC_Common_TypeDef * ADCxy_COMMON, uint32_t MultiTwoSamplingDelay)

Function description

Set ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **MultiTwoSamplingDelay:** This parameter can be one of the following values:
  – LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  – LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES
  – LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES_8_BITS
  – LL_ADC_MULTI_TWOSMP_DELAY_14CYCLES (1)
  – LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES (1)
  – LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES_10_BITS
  – LL_ADC_MULTI_TWOSMP_DELAY_16CYCLES (2)

  (1) Parameter available only if ADC resolution is 14, 12 or 10 bits. (2) Parameter available only if ADC resolution is 14 or 12 bits.

Return values

- **None:**

Notes

- The sampling delay range depends on ADC resolution: ADC resolution 14 bits can have maximum delay of 16 cycles.ADC resolution 12 bits can have maximum delay of 16 cycles.ADC resolution 10 bits can have maximum delay of 15 cycles.ADC resolution 8 bits can have maximum delay of 13 cycles.
- On this STM32 series, setting of this feature is conditioned to ADC state: All ADC instances of the ADC common group must be disabled. This check can be done with function LL_ADC_IsEnabled() for each ADC instance or by using helper macro helper macro __LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE().

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_SetMultiTwoSamplingDelay

LL_ADC_GetMultiTwoSamplingDelay

Function name

__STATIC_INLINE uint32_t LL_ADC_GetMultiTwoSamplingDelay (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

Get ADC multimode delay between 2 sampling phases.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE
  - LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES
  - LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES_8_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_14CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES (1)
  - LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES_10_BITS
  - LL_ADC_MULTI_TWOSMP_DELAY_16CYCLES (2)

  (1) Parameter available only if ADC resolution is 14, 12 or 10 bits. (2) Parameter available only if ADC resolution is 14 or 12 bits.

Reference Manual to LL API cross reference:

- CCR DELAY LL_ADC_GetMultiTwoSamplingDelay

LL_ADC_EnableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_EnableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Put ADC instance in deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_EnableDeepPowerDown

LL_ADC_DisableDeepPowerDown

Function name

__STATIC_INLINE void LL_ADC_DisableDeepPowerDown (ADC_TypeDef * ADCx)

Function description

Disable ADC deep power down mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- In case of ADC calibration necessary: When ADC is in deep-power-down state, the internal analog calibration is lost. After exiting from deep power down, calibration must be relaunched or calibration factor (preliminarily saved) must be set back into calibration register.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_DisableDeepPowerDown

LL_ADC_IsDeepPowerDownEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDeepPowerDownEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance deep power down state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** deep power down is disabled, 1: deep power down is enabled.

Reference Manual to LL API cross reference:

- CR DEEPPWD LL_ADC_IsDeepPowerDownEnabled

LL_ADC_EnableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Enable ADC instance internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC internal voltage regulator enable, a delay for ADC internal voltage regulator stabilization is required before performing a ADC calibration or ADC enable. Refer to device datasheet, parameter tADCVREG_STUP. Refer to literal LL_ADC_DELAY_INTERNAL_REGUL_STAB_US.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Disable ADC internal voltage regulator.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance internal voltage regulator state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** internal regulator is disabled, 1: internal regulator is enabled.

Reference Manual to LL API cross reference:

- CR ADVREGEN LL_ADC_IsInternalRegulatorEnabled

LL_ADC_Enable

Function name

__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description

Enable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, after ADC enable, a delay for ADC internal analog stabilization is required before performing a ADC conversion start. Refer to device datasheet, parameter tSTAB.
- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled and ADC internal voltage regulator enabled.

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_Enable

LL_ADC_Disable

Function name

__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)

Function description

Disable the selected ADC instance.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be not disabled. Must be enabled without conversion on going on either groups regular or injected.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_Disable

LL_ADC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabled (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance enable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC is disabled, 1: ADC is enabled.

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- CR ADEN LL_ADC_IsEnabled

LL_ADC_IsDisableOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDisableOngoing (const ADC_TypeDef * ADCx)

Function description

Get the selected ADC instance disable state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADDIS LL_ADC_IsDisableOngoing

LL_ADC_StartCalibration

Function name

__STATIC_INLINE void LL_ADC_StartCalibration (ADC_TypeDef * ADCx, uint32_t CalibrationMode)

Function description

Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance
- **CalibrationMode:** This parameter can be one of the following values:
  - LL_ADC_CALIB_OFFSET
  - LL_ADC_CALIB_OFFSET_LINEARITY

Return values

- **None:**

Notes

- On this STM32 series, a minimum number of ADC clock cycles are required between ADC end of calibration and ADC enable. Refer to literal LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES.
- Calibration duration: Calibration of offset: 520 ADC clock cyclesCalibration of linearity: 131072 ADC clock cycles
- For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes (calibration run must be performed for each of these differential modes, if used afterwards and if the application requires their calibration). Calibration of linearity is common to both single-ended and differential modes (calibration run can be performed only once).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be ADC disabled.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_StartCalibration
- CR ADCALDIF LL_ADC_StartCalibration
- CR ADCALLIN LL_ADC_StartCalibration

LL_ADC_IsCalibrationOnGoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsCalibrationOnGoing (const ADC_TypeDef * ADCx)

Function description

Get ADC calibration state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** calibration complete, 1: calibration in progress.

Reference Manual to LL API cross reference:

- CR ADCAL LL_ADC_IsCalibrationOnGoing

LL_ADC_REG_StartConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group regular, without conversion stop command on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_StartConversion

LL_ADC_REG_StopConversion

Function name

__STATIC_INLINE void LL_ADC_REG_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group regular conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group regular, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_StopConversion

LL_ADC_REG_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTART LL_ADC_REG_IsConversionOngoing

LL_ADC_REG_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group regular.

Reference Manual to LL API cross reference:

- CR ADSTP LL_ADC_REG_IsStopConversionOngoing

LL_ADC_REG_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadConversionData32 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData32

LL_ADC_REG_ReadConversionData16

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData16 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 16 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData16

LL_ADC_REG_ReadConversionData14

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData14 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 14 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData14

LL_ADC_REG_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData12 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData12

LL_ADC_REG_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_REG_ReadConversionData10 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData10

LL_ADC_REG_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData8 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_REG_ReadConversionData32.

Reference Manual to LL API cross reference:

- DR RDATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadMultiConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_REG_ReadMultiConversionData32 (const ADC_Common_TypeDef * ADCxy_COMMON, uint32_t ConversionData)

Function description

Get ADC multimode conversion data of ADC master, ADC slave or raw data with ADC master and slave concatenated.

Parameters

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ConversionData:** This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
  - LL_ADC_MULTI_MASTER_SLAVE

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes

- If raw data with ADC master and slave concatenated is retrieved, a macro is available to get the conversion data of ADC master or ADC slave: see helper macro __LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE(). (however this macro is mainly intended for multimode transfer by DMA, because this function can do the same by getting multimode conversion data of ADC master or ADC slave separately).

Reference Manual to LL API cross reference:

- CDR RDATA_MST LL_ADC_REG_ReadMultiConversionData32
- CDR RDATA_SLV LL_ADC_REG_ReadMultiConversionData32

LL_ADC_INJ_StartConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StartConversion (ADC_TypeDef * ADCx)

Function description

Start ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, this function is relevant for both internal trigger (SW start) and external trigger: If ADC trigger has been set to software start, ADC conversion starts immediately.If ADC trigger has been set to external trigger, ADC conversion will start at next trigger event (on the selected trigger edge) following the ADC start conversion command.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled without conversion on going on group injected, without conversion stop command on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_StartConversion

LL_ADC_INJ_StopConversion

Function name

__STATIC_INLINE void LL_ADC_INJ_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop ADC group injected conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be enabled with conversion on going on group injected, without ADC disable command on going.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_StopConversion

LL_ADC_INJ_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected conversion state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no conversion is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTART LL_ADC_INJ_IsConversionOngoing

LL_ADC_INJ_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Get ADC group injected command of conversion stop state.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** no command of conversion stop is on going on ADC group injected.

Reference Manual to LL API cross reference:

- CR JADSTP LL_ADC_INJ_IsStopConversionOngoing

LL_ADC_INJ_ReadConversionData32

Function name

__STATIC_INLINE uint32_t LL_ADC_INJ_ReadConversionData32 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group regular conversion data, range fit for all ADC configurations: all ADC resolutions and all oversampling increased data width (for devices with feature oversampling).

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData32
- JDR2 JDATA LL_ADC_INJ_ReadConversionData32
- JDR3 JDATA LL_ADC_INJ_ReadConversionData32
- JDR4 JDATA LL_ADC_INJ_ReadConversionData32

LL_ADC_INJ_ReadConversionData16

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData16 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 16 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData16
- JDR2 JDATA LL_ADC_INJ_ReadConversionData16
- JDR3 JDATA LL_ADC_INJ_ReadConversionData16
- JDR4 JDATA LL_ADC_INJ_ReadConversionData16

LL_ADC_INJ_ReadConversionData14

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData14 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 14 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_INJ_RANK_1
  - LL_ADC_INJ_RANK_2
  - LL_ADC_INJ_RANK_3
  - LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData14
- JDR2 JDATA LL_ADC_INJ_ReadConversionData14
- JDR3 JDATA LL_ADC_INJ_ReadConversionData14
- JDR4 JDATA LL_ADC_INJ_ReadConversionData14

LL_ADC_INJ_ReadConversionData12

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData12 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 12 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData12
- JDR2 JDATA LL_ADC_INJ_ReadConversionData12
- JDR3 JDATA LL_ADC_INJ_ReadConversionData12
- JDR4 JDATA LL_ADC_INJ_ReadConversionData12

LL_ADC_INJ_ReadConversionData10

Function name

__STATIC_INLINE uint16_t LL_ADC_INJ_ReadConversionData10 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 10 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData10
- JDR2 JDATA LL_ADC_INJ_ReadConversionData10
- JDR3 JDATA LL_ADC_INJ_ReadConversionData10
- JDR4 JDATA LL_ADC_INJ_ReadConversionData10

LL_ADC_INJ_ReadConversionData8

Function name

__STATIC_INLINE uint8_t LL_ADC_INJ_ReadConversionData8 (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC group injected conversion data, range fit for ADC resolution 8 bits.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  – LL_ADC_INJ_RANK_1
  – LL_ADC_INJ_RANK_2
  – LL_ADC_INJ_RANK_3
  – LL_ADC_INJ_RANK_4

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- For devices with feature oversampling: Oversampling can increase data width, function for extended range may be needed: LL_ADC_INJ_ReadConversionData32.

Reference Manual to LL API cross reference:

- JDR1 JDATA LL_ADC_INJ_ReadConversionData8
- JDR2 JDATA LL_ADC_INJ_ReadConversionData8
- JDR3 JDATA LL_ADC_INJ_ReadConversionData8
- JDR4 JDATA LL_ADC_INJ_ReadConversionData8

LL_ADC_IsActiveFlag_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_ADRDY (const ADC_TypeDef * ADCx)

Function description

Get flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_IsActiveFlag_ADRDY

LL_ADC_IsActiveFlag_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_IsActiveFlag_EOC

LL_ADC_IsActiveFlag_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_IsActiveFlag_EOS

LL_ADC_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVR (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_IsActiveFlag_OVR

LL_ADC_IsActiveFlag_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOSMP (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_IsActiveFlag_EOSMP

LL_ADC_IsActiveFlag_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOC (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_IsActiveFlag_JEOC

LL_ADC_IsActiveFlag_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_JEOS (const ADC_TypeDef * ADCx)

Function description

Get flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_IsActiveFlag_JEOS

LL_ADC_IsActiveFlag_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD1 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 1 flag.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_IsActiveFlag_AWD1

LL_ADC_IsActiveFlag_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD2 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_IsActiveFlag_AWD2

LL_ADC_IsActiveFlag_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD3 (const ADC_TypeDef * ADCx)

Function description

Get flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_IsActiveFlag_AWD3

LL_ADC_ClearFlag_ADRDY

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_ADRDY (ADC_TypeDef * ADCx)

Function description

Clear flag ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- On this STM32 series, flag LL_ADC_FLAG_ADRDY is raised when the ADC is enabled and when conversion clock is active. (not only core clock: this ADC has a dual clock domain)

Reference Manual to LL API cross reference:

- ISR ADRDY LL_ADC_ClearFlag_ADRDY

LL_ADC_ClearFlag_EOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOC LL_ADC_ClearFlag_EOC

LL_ADC_ClearFlag_EOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOS LL_ADC_ClearFlag_EOS

LL_ADC_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_OVR (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR OVR LL_ADC_ClearFlag_OVR

LL_ADC_ClearFlag_EOSMP

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOSMP (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group regular end of sampling phase.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOSMP LL_ADC_ClearFlag_EOSMP

LL_ADC_ClearFlag_JEOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOC (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOC LL_ADC_ClearFlag_JEOC

LL_ADC_ClearFlag_JEOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_JEOS (ADC_TypeDef * ADCx)

Function description

Clear flag ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR JEOS LL_ADC_ClearFlag_JEOS

LL_ADC_ClearFlag_AWD1

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD1 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD1 LL_ADC_ClearFlag_AWD1

LL_ADC_ClearFlag_AWD2

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD2 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD2 LL_ADC_ClearFlag_AWD2

LL_ADC_ClearFlag_AWD3

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD3 (ADC_TypeDef * ADCx)

Function description

Clear flag ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR AWD3 LL_ADC_ClearFlag_AWD3

LL_ADC_EnableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_EnableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Enable ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_EnableIT_ADRDY

LL_ADC_EnableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_EnableIT_EOC

LL_ADC_EnableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_EnableIT_EOS

LL_ADC_EnableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_EnableIT_OVR (ADC_TypeDef * ADCx)

Function description

Enable ADC group regular interruption overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_EnableIT_OVR

LL_ADC_EnableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_EnableIT_EOSMP

LL_ADC_EnableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_EnableIT_JEOC

LL_ADC_EnableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_EnableIT_JEOS

LL_ADC_EnableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_EnableIT_AWD1

LL_ADC_EnableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_EnableIT_AWD2

LL_ADC_EnableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_EnableIT_AWD3

LL_ADC_EnableIT_EOCAL

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOCAL (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC end of calibration.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCALIE LL_ADC_EnableIT_EOCAL

LL_ADC_DisableIT_ADRDY

Function name

__STATIC_INLINE void LL_ADC_DisableIT_ADRDY (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC ready.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_DisableIT_ADRDY

LL_ADC_DisableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_DisableIT_EOC

LL_ADC_DisableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_DisableIT_EOS

LL_ADC_DisableIT_OVR

Function name

__STATIC_INLINE void LL_ADC_DisableIT_OVR (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular overrun.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_DisableIT_OVR

LL_ADC_DisableIT_EOSMP

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOSMP (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of sampling.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_DisableIT_EOSMP

LL_ADC_DisableIT_JEOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOC (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group regular end of unitary conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_DisableIT_JEOC

LL_ADC_DisableIT_JEOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_JEOS (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC group injected end of sequence conversions.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_DisableIT_JEOS

LL_ADC_DisableIT_AWD1

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD1 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 1.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_DisableIT_AWD1

LL_ADC_DisableIT_AWD2

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD2 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 2.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_DisableIT_AWD2

LL_ADC_DisableIT_AWD3

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD3 (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC analog watchdog 3.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_DisableIT_AWD3

LL_ADC_DisableIT_EOCAL

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOCAL (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC end of calibration.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOCALIE LL_ADC_DisableIT_EOCAL

LL_ADC_IsEnabledIT_ADRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_ADRDY (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC ready (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_IsEnabledIT_ADRDY

LL_ADC_IsEnabledIT_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOCIE LL_ADC_IsEnabledIT_EOC

LL_ADC_IsEnabledIT_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSIE LL_ADC_IsEnabledIT_EOS

LL_ADC_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_OVR (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular overrun (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER OVRIE LL_ADC_IsEnabledIT_OVR

LL_ADC_IsEnabledIT_EOSMP

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOSMP (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group regular end of sampling (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOSMPIE LL_ADC_IsEnabledIT_EOSMP

LL_ADC_IsEnabledIT_JEOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOC (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of unitary conversion (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOCIE LL_ADC_IsEnabledIT_JEOC

LL_ADC_IsEnabledIT_JEOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_JEOS (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC group injected end of sequence conversions (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER JEOSIE LL_ADC_IsEnabledIT_JEOS

LL_ADC_IsEnabledIT_AWD1

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD1 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC analog watchdog 1 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD1IE LL_ADC_IsEnabledIT_AWD1

LL_ADC_IsEnabledIT_AWD2

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD2 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 2 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD2IE LL_ADC_IsEnabledIT_AWD2

LL_ADC_IsEnabledIT_AWD3

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD3 (const ADC_TypeDef * ADCx)

Function description

Get state of interruption Get ADC analog watchdog 3 (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER AWD3IE LL_ADC_IsEnabledIT_AWD3

LL_ADC_IsEnabledIT_EOCAL

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOCAL (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC end of calibration (0: interrupt disabled, 1: interrupt enabled).

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER EOCALIE LL_ADC_IsEnabledIT_EOCAL

LL_ADC_CommonDeInit

Function name

ErrorStatus LL_ADC_CommonDeInit (ADC_Common_TypeDef * pADCxy_COMMON)

Function description

De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.

Parameters

- **pADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are de-initialized
  - ERROR: not applicable

Notes

- This function is performing a hard reset, using high level clock source RCC ADC reset. Caution: On this STM32 series, if several ADC instances are available on the selected device, RCC ADC reset will reset all ADC instances belonging to the common ADC instance. To de-initialize only 1 ADC instance, use function LL_ADC_DeInit().

LL_ADC_CommonInit

Function name

ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * pADCxy_COMMON, LL_ADC_CommonInitTypeDef * pADC_CommonInitStruct)

Function description

Initialize some features of ADC common parameters (all ADC instances belonging to the same ADC common instance) and multimode (for devices with several ADC instances available).

Parameters

- **pADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **pADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC common registers are initialized
  - ERROR: ADC common registers are not initialized

Notes

- The setting of ADC common parameters is conditioned to ADC instances state: All ADC instances belonging to the same ADC common instance must be disabled.

LL_ADC_CommonStructInit

Function name

void LL_ADC_CommonStructInit (LL_ADC_CommonInitTypeDef * pADC_CommonInitStruct)

Function description

Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters

- **pADC_CommonInitStruct:** Pointer to a LL_ADC_CommonInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_DeInit

Function name

ErrorStatus LL_ADC_DeInit (ADC_TypeDef * pADCx)

Function description

De-initialize registers of the selected ADC instance to their default reset values.

Parameters

- **pADCx:** ADC instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: ADC registers are de-initialized
  – ERROR: ADC registers are not de-initialized

Notes

- To reset all ADC instances quickly (perform a hard reset), use function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name

ErrorStatus LL_ADC_Init (ADC_TypeDef * pADCx, LL_ADC_InitTypeDef * pADC_InitStruct)

Function description

Initialize some features of ADC instance.

Parameters

- **pADCx:** ADC instance
- **pADC_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: ADC registers are initialized
  – ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC instance. Affects both group regular and group injected (availability of ADC group injected depends on STM32 families). Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: ADC instance .
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, some other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name

void LL_ADC_StructInit (const ADC_TypeDef * pADCx, LL_ADC_InitTypeDef * pADC_InitStruct)

Function description

Set each LL_ADC_InitTypeDef field to default value.

Parameters

- **pADCx:** ADC instance
- **pADC_InitStruct:** Pointer to a LL_ADC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_REG_Init

Function name

ErrorStatus LL_ADC_REG_Init (ADC_TypeDef * pADCx, LL_ADC_REG_InitTypeDef * pADC_RegInitStruct)

Function description

Initialize some features of ADC group regular.

Parameters

- **pADCx:** ADC instance
- **pADC_RegInitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: ADC registers are initialized
  – ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group regular. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "REG").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group regular or group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name

void LL_ADC_REG_StructInit (const ADC_TypeDef * pADCx, LL_ADC_REG_InitTypeDef * pADC_RegInitStruct)

Function description

Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters

- **pADCx:** ADC instance
- **pADC_RegInitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_ADC_INJ_Init

Function name

ErrorStatus LL_ADC_INJ_Init (ADC_TypeDef * pADCx, LL_ADC_INJ_InitTypeDef * pADC_InjInitStruct)

Function description

Initialize some features of ADC group injected.

Parameters

- **pADCx:** ADC instance
- **pADC_InjInitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ADC registers are initialized
  - ERROR: ADC registers are not initialized

Notes

- These parameters have an impact on ADC scope: ADC group injected. Refer to corresponding unitary functions into Configuration of ADC hierarchical scope: group regular (functions with prefix "INJ").
- The setting of these parameters by function LL_ADC_Init() is conditioned to ADC state: ADC instance must be disabled. This condition is applied to all ADC features, for efficiency and compatibility over all STM32 families. However, the different features can be set under different ADC state conditions (setting possible with ADC enabled without conversion on going, ADC enabled with conversion on going, ...) Each feature can be updated afterwards with a unitary function and potentially with ADC in a different state than disabled, refer to description of each function for setting conditioned to ADC state.
- After using this function, other features must be configured using LL unitary functions. The minimum configuration remaining to be done is: Set ADC group injected sequencer: map channel on the selected sequencer rank. Refer to function LL_ADC_INJ_SetSequencerRanks().Set ADC channel sampling time Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_INJ_StructInit

Function name

void LL_ADC_INJ_StructInit (LL_ADC_INJ_InitTypeDef * pADC_InjInitStruct)

Function description

Set each LL_ADC_INJ_InitTypeDef field to default value.

Parameters

- **pADC_InjInitStruct:** Pointer to a LL_ADC_INJ_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

90.3 ADC Firmware driver defines

The following section lists the various define and macros of the module.

90.3.1 ADC

ADC

ADC instance - Autonomous deep power down mode

LL_ADC_LP_AUTONOMOUS_DPD_DISABLE

ADC deep power down in autonomous mode disabled

LL_ADC_LP_AUTONOMOUS_DPD_ENABLE

ADC deep power down in autonomous mode enabled

ADC instance - Low power mode auto power-off

LL_ADC_LP_AUTOPOWEROFF_DISABLE

ADC low power mode auto power-off disabled

LL_ADC_LP_AUTOPOWEROFF_ENABLE

ADC low power mode auto power-off enabled: the ADC automatically powers-off after a ADC conversion and automatically wakes up when a new ADC conversion is triggered (with startup time between trigger and start of sampling). See description with function

Analog watchdog - Monitored channels

LL_ADC_AWD_DISABLE

ADC analog watchdog monitoring disabled

LL_ADC_AWD_ALL_CHANNELS_REG

ADC analog watchdog monitoring of all channels, converted by group regular only

LL_ADC_AWD_ALL_CHANNELS_INJ

ADC analog watchdog monitoring of all channels, converted by group injected only

LL_ADC_AWD_ALL_CHANNELS_REG_INJ

ADC analog watchdog monitoring of all channels, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_0_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group regular only

LL_ADC_AWD_CHANNEL_0_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by group injected only

LL_ADC_AWD_CHANNEL_0_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN0, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_1_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group regular only

LL_ADC_AWD_CHANNEL_1_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by group injected only

LL_ADC_AWD_CHANNEL_1_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN1, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_2_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group regular only

LL_ADC_AWD_CHANNEL_2_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by group injected only

LL_ADC_AWD_CHANNEL_2_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN2, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_3_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group regular only

LL_ADC_AWD_CHANNEL_3_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by group injected only

LL_ADC_AWD_CHANNEL_3_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN3, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_4_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group regular only

LL_ADC_AWD_CHANNEL_4_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by group injected only

LL_ADC_AWD_CHANNEL_4_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN4, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_5_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group regular only

LL_ADC_AWD_CHANNEL_5_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by group injected only

LL_ADC_AWD_CHANNEL_5_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN5, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_6_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group regular only

LL_ADC_AWD_CHANNEL_6_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by group injected only

LL_ADC_AWD_CHANNEL_6_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN6, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_7_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group regular only

LL_ADC_AWD_CHANNEL_7_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by group injected only

LL_ADC_AWD_CHANNEL_7_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN7, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_8_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group regular only

LL_ADC_AWD_CHANNEL_8_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by group injected only

LL_ADC_AWD_CHANNEL_8_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN8, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_9_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group regular only

LL_ADC_AWD_CHANNEL_9_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by group injected only

LL_ADC_AWD_CHANNEL_9_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN9, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_10_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group regular only

LL_ADC_AWD_CHANNEL_10_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by group injected only

LL_ADC_AWD_CHANNEL_10_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN10, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_11_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group regular only

LL_ADC_AWD_CHANNEL_11_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by group injected only

LL_ADC_AWD_CHANNEL_11_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN11, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_12_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group regular only

LL_ADC_AWD_CHANNEL_12_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by group injected only

LL_ADC_AWD_CHANNEL_12_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN12, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_13_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group regular only

LL_ADC_AWD_CHANNEL_13_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by group injected only

LL_ADC_AWD_CHANNEL_13_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN13, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_14_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group regular only

LL_ADC_AWD_CHANNEL_14_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by group injected only

LL_ADC_AWD_CHANNEL_14_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN14, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_15_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group regular only

LL_ADC_AWD_CHANNEL_15_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by group injected only

LL_ADC_AWD_CHANNEL_15_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN15, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_16_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group regular only

LL_ADC_AWD_CHANNEL_16_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by group injected only

LL_ADC_AWD_CHANNEL_16_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN16, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_17_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group regular only

LL_ADC_AWD_CHANNEL_17_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by group injected only

LL_ADC_AWD_CHANNEL_17_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN17, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_18_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group regular only

LL_ADC_AWD_CHANNEL_18_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by group injected only

LL_ADC_AWD_CHANNEL_18_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN18, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_19_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_19_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_19_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_20_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_20_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_20_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_21_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_21_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_21_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_22_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_22_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_22_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CHANNEL_23_REG

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_23_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by group injected only

LL_ADC_AWD_CHANNEL_23_REG_INJ

ADC analog watchdog monitoring of ADC external channel (channel connected to GPIO pin) ADCx_IN19, converted by either group regular or injected

LL_ADC_AWD_CH_VREFINT_REG

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only

LL_ADC_AWD_CH_VREFINT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only

LL_ADC_AWD_CH_VREFINT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected

LL_ADC_AWD_CH_TEMPSENSOR_REG

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only

LL_ADC_AWD_CH_TEMPSENSOR_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group injected only

LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by either group regular or injected

LL_ADC_AWD_CH_VBAT_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, converted by group regular only

LL_ADC_AWD_CH_VBAT_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, converted by group injected only

LL_ADC_AWD_CH_VBAT_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda

LL_ADC_AWD_CH_VCORE_REG

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group regular only

LL_ADC_AWD_CH_VCORE_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by group injected only

LL_ADC_AWD_CH_VCORE_REG_INJ

ADC analog watchdog monitoring of ADC internal channel connected to VrefInt: Internal voltage reference, converted by either group regular or injected

LL_ADC_AWD_CH_DAC1CH1_ADC4_REG

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 1, channel specific to ADC4, converted by group regular only

LL_ADC_AWD_CH_DAC1CH2_ADC4_REG

ADC analog watchdog monitoring of ADC internal channel connected to DAC1 channel 2, channel specific to ADC4, converted by group regular only

LL_ADC_AWD_CH_TEMPSENSOR_ADC4_REG

ADC analog watchdog monitoring of ADC internal channel connected to emperature sensor, channel specific to ADC4, converted by group regular only

LL_ADC_AWD_CH_VBAT_ADC4_REG

ADC analog watchdog monitoring of ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda, channel specific to ADC4, converted by group regular only

Analog watchdog - filtering config

LL_ADC_AWD_FILTERING_NONE

ADC analog watchdog no filtering, one out-of-window sample is needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_2SAMPLES

ADC analog watchdog 2 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_3SAMPLES

ADC analog watchdog 3 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_4SAMPLES

ADC analog watchdog 4 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_5SAMPLES

ADC analog watchdog 5 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_6SAMPLES

ADC analog watchdog 6 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_7SAMPLES

ADC analog watchdog 7 consecutives out-of-window samples are needed to raise flag or interrupt

LL_ADC_AWD_FILTERING_8SAMPLES

ADC analog watchdog 8 consecutives out-of-window samples are needed to raise flag or interrupt

Analog watchdog - Analog watchdog number

LL_ADC_AWD1

ADC analog watchdog number 1

LL_ADC_AWD2

ADC analog watchdog number 2

LL_ADC_AWD3

ADC analog watchdog number 3

Analog watchdog - Thresholds

LL_ADC_AWD_THRESHOLD_HIGH

ADC analog watchdog threshold high

LL_ADC_AWD_THRESHOLD_LOW

ADC analog watchdog threshold low

ADC indexes for linear calibration

LL_ADC_CALIB_OFFSET_INDEX

Offset Calibration Index

LL_ADC_CALIB_LINEARITY_INDEX1

Linearity Calibration Index 1

LL_ADC_CALIB_LINEARITY_INDEX2

Linearity Calibration Index 2

LL_ADC_CALIB_LINEARITY_INDEX3

Linearity Calibration Index 3

LL_ADC_CALIB_LINEARITY_INDEX4

Linearity Calibration Index 4

LL_ADC_CALIB_LINEARITY_INDEX5

Linearity Calibration Index 5

LL_ADC_CALIB_LINEARITY_INDEX6

Linearity Calibration Index 6

LL_ADC_CALIB_LINEARITY_INDEX7

Linearity Calibration Index 7

LL_ADC_CALIB_INTEROFFSET_INDEX

Linearity Calibration Index 8

ADC instance - Calibration mode for offset and linearity

LL_ADC_CALIB_OFFSET

Calibration of ADC offset. Duration of calibration of offset duration: 1280 ADC clock cycles. For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes.

LL_ADC_CALIB_LINEARITY

Calibration of ADC linearity. Duration of calibration of linearity: 15104 ADC clock cycles. For devices with differential mode available: Calibration of linearity is common to both single-ended and differential modes.

LL_ADC_CALIB_OFFSET_LINEARITY

Calibration of ADC offset and linearity. Duration of calibration of offset and linearity: 16384 ADC clock cycles. For devices with differential mode available: Calibration of offset is specific to each of single-ended and differential modes, calibration of linearity is common to both single-ended and differential modes.

ADC instance - Channel number

LL_ADC_CHANNEL_0

ADC external channel (channel connected to GPIO pin) ADCx_IN0

LL_ADC_CHANNEL_1

ADC external channel (channel connected to GPIO pin) ADCx_IN1

LL_ADC_CHANNEL_2

ADC external channel (channel connected to GPIO pin) ADCx_IN2

LL_ADC_CHANNEL_3

ADC external channel (channel connected to GPIO pin) ADCx_IN3

LL_ADC_CHANNEL_4

ADC external channel (channel connected to GPIO pin) ADCx_IN4

LL_ADC_CHANNEL_5
ADC external channel (channel connected to GPIO pin) ADCx_IN5

LL_ADC_CHANNEL_6
ADC external channel (channel connected to GPIO pin) ADCx_IN6

LL_ADC_CHANNEL_7
ADC external channel (channel connected to GPIO pin) ADCx_IN7

LL_ADC_CHANNEL_8
ADC external channel (channel connected to GPIO pin) ADCx_IN8

LL_ADC_CHANNEL_9
ADC external channel (channel connected to GPIO pin) ADCx_IN9

LL_ADC_CHANNEL_10
ADC external channel (channel connected to GPIO pin) ADCx_IN10

LL_ADC_CHANNEL_11
ADC external channel (channel connected to GPIO pin) ADCx_IN11

LL_ADC_CHANNEL_12
ADC external channel (channel connected to GPIO pin) ADCx_IN12

LL_ADC_CHANNEL_13
ADC external channel (channel connected to GPIO pin) ADCx_IN13

LL_ADC_CHANNEL_14
ADC external channel (channel connected to GPIO pin) ADCx_IN14

LL_ADC_CHANNEL_15
ADC external channel (channel connected to GPIO pin) ADCx_IN15

LL_ADC_CHANNEL_16
ADC external channel (channel connected to GPIO pin) ADCx_IN16

LL_ADC_CHANNEL_17
ADC external channel (channel connected to GPIO pin) ADCx_IN17

LL_ADC_CHANNEL_18
ADC external channel (channel connected to GPIO pin) ADCx_IN18

LL_ADC_CHANNEL_19
ADC external channel (channel connected to GPIO pin) ADCx_IN19

LL_ADC_CHANNEL_20
ADC external channel (channel connected to GPIO pin) ADCx_IN20

LL_ADC_CHANNEL_21
ADC external channel (channel connected to GPIO pin) ADCx_IN21

LL_ADC_CHANNEL_22
ADC external channel (channel connected to GPIO pin) ADCx_IN22

LL_ADC_CHANNEL_23
ADC external channel (channel connected to GPIO pin) ADCx_IN23

LL_ADC_CHANNEL_VREFINT

ADC internal channel connected to VrefInt: Internal voltage reference. .

LL_ADC_CHANNEL_TEMPSENSOR

ADC internal channel connected to Temperature sensor.

LL_ADC_CHANNEL_VBAT

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda.

LL_ADC_CHANNEL_VCORE

ADC internal channel connected to Vcore. On STM32U5, ADC channel available only on ADC instance: ADC4.

LL_ADC_CHANNEL_TEMPSENSOR_ADC4

ADC internal channel connected to Temperature sensor. On STM32U5, ADC channel available only on ADC instance: ADC4.

LL_ADC_CHANNEL_VBAT_ADC4

ADC internal channel connected to Vbat/4: Vbat voltage through a divider ladder of factor 1/4 to have Vbat always below Vdda. On STM32U5, ADC channel available only on ADC instance: ADC4.

LL_ADC_CHANNEL_DAC1CH1_ADC4

ADC internal channel connected to DAC1 channel 1, channel specific to ADC4

LL_ADC_CHANNEL_DAC1CH2_ADC4

ADC internal channel connected to DAC1 channel 2, channel specific to ADC4

LL_ADC_CHANNEL_DIFFERENCIATION_VREFINT_VCORE

Dummy bit for driver internal usage, not used in ADC channel setting registers CHSELR or SQRx

Channel - Sampling time

LL_ADC_SAMPLINGTIME_5CYCLES

Sampling time 5 ADC clock cycles

LL_ADC_SAMPLINGTIME_6CYCLES

Sampling time 6 ADC clock cycles

LL_ADC_SAMPLINGTIME_12CYCLES

Sampling time 12 ADC clock cycles

LL_ADC_SAMPLINGTIME_20CYCLES

Sampling time 20 ADC clock cycles

LL_ADC_SAMPLINGTIME_36CYCLES

Sampling time 36 ADC clock cycles

LL_ADC_SAMPLINGTIME_68CYCLES

Sampling time 68 ADC clock cycles

LL_ADC_SAMPLINGTIME_391CYCLES

Sampling time 391 ADC clock cycles

LL_ADC_SAMPLINGTIME_814CYCLES

Sampling time 814 ADC clock cycles

LL_ADC_SAMPLINGTIME_5CYCLE

Keep old definition for compatibility

LL_ADC_SAMPLINGTIME_391CYCLES_5

Keep old definition for compatibility

LL_ADC4_SAMPLINGTIME_1CYCLE_5

Sampling time 1.5 ADC clock cycle

LL_ADC4_SAMPLINGTIME_3CYCLES_5

Sampling time 3.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_7CYCLES_5

Sampling time 7.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_19CYCLES_5

Sampling time 19.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_39CYCLES_5

Sampling time 39.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_79CYCLES_5

Sampling time 79.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_814CYCLES_5

Sampling time 814.5 ADC clock cycles

LL_ADC4_SAMPLINGTIME_160CYCLES_5

Keep old definition for compatibility

Channel - Single or differential ending

LL_ADC_SINGLE_ENDED

ADC channel ending set to single ended (literal also used to set calibration mode)

LL_ADC_DIFFERENTIAL_ENDED

ADC channel ending set to differential (literal also used to set calibration mode)

LL_ADC_BOTH_SINGLE_DIFF_ENDED

ADC channel ending set to both single ended and differential (literal used only to set calibration factors)

ADC common - Clock source

LL_ADC_CLOCK_ASYNC_DIV1

ADC asynchronous clock without prescaler

LL_ADC_CLOCK_ASYNC_DIV2

ADC asynchronous clock with prescaler division by 2

LL_ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4

LL_ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6

LL_ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8

LL_ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10

LL_ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12

LL_ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16

LL_ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32

LL_ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64

LL_ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128

LL_ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256

ADC common - Measurement path to internal channels

LL_ADC_PATH_INTERNAL_NONE

ADC measurement paths all disabled

LL_ADC_PATH_INTERNAL_VREFINT

ADC measurement path to internal channel VrefInt

LL_ADC_PATH_INTERNAL_TEMPSENSOR

ADC measurement path to internal channel temperature sensor

LL_ADC_PATH_INTERNAL_VBAT

ADC measurement path to internal channel Vbat

LL_ADC_PATH_INTERNAL_VCORE

ADC measurement path to internal channel VCore

ADC instance - Data alignment

LL_ADC_DATA_ALIGN_RIGHT

ADC conversion data alignment: right aligned (alignment on data register LSB bit 0)

LL_ADC_DATA_ALIGN_LEFT

ADC conversion data alignment: left aligned (alignment on data register MSB bit 15)

ADC flags

LL_ADC_FLAG_ADRDY

ADC flag ADC instance ready

LL_ADC_FLAG_EOC

ADC flag ADC group regular end of unitary conversion

LL_ADC_FLAG_EOS

ADC flag ADC group regular end of sequence conversions

LL_ADC_FLAG_OVR

ADC flag ADC group regular overrun

LL_ADC_FLAG_EOSMP

ADC flag ADC group regular end of sampling phase

LL_ADC_FLAG_JEOC

ADC flag ADC group injected end of unitary conversion

LL_ADC_FLAG_JEOS

ADC flag ADC group injected end of sequence conversions

LL_ADC_FLAG_AWD1

ADC flag ADC analog watchdog 1

LL_ADC_FLAG_AWD2

ADC flag ADC analog watchdog 2

LL_ADC_FLAG_AWD3

ADC flag ADC analog watchdog 3

LL_ADC_FLAG_EOCAL

ADC flag end of calibration

LL_ADC_FLAG_LDORDY

ADC flag Voltage Regulator (LDO) Ready

LL_ADC_FLAG_ADRDY_MST

ADC flag ADC multimode master instance ready

LL_ADC_FLAG_ADRDY_SLV

ADC flag ADC multimode slave instance ready

LL_ADC_FLAG_EOC_MST

ADC flag ADC multimode master group regular end of unitary conversion

LL_ADC_FLAG_EOC_SLV

ADC flag ADC multimode slave group regular end of unitary conversion

LL_ADC_FLAG_EOS_MST

ADC flag ADC multimode master group regular end of sequence conversions

LL_ADC_FLAG_EOS_SLV

ADC flag ADC multimode slave group regular end of sequence conversions

LL_ADC_FLAG_OVR_MST

ADC flag ADC multimode master group regular overrun

LL_ADC_FLAG_OVR_SLV

ADC flag ADC multimode slave group regular overrun

LL_ADC_FLAG_EOSMP_MST

ADC flag ADC multimode master group regular end of sampling phase

LL_ADC_FLAG_EOSMP_SLV

ADC flag ADC multimode slave group regular end of sampling phase

LL_ADC_FLAG_JEOC_MST

ADC flag ADC multimode master group injected end of unitary conversion

LL_ADC_FLAG_JEOC_SLV

ADC flag ADC multimode slave group injected end of unitary conversion

LL_ADC_FLAG_JEOS_MST

ADC flag ADC multimode master group injected end of sequence conversions

LL_ADC_FLAG_JEOS_SLV

ADC flag ADC multimode slave group injected end of sequence conversions

LL_ADC_FLAG_AWD1_MST

ADC flag ADC multimode master analog watchdog 1 of the ADC master

LL_ADC_FLAG_AWD1_SLV

ADC flag ADC multimode slave analog watchdog 1 of the ADC slave

LL_ADC_FLAG_AWD2_MST

ADC flag ADC multimode master analog watchdog 2 of the ADC master

LL_ADC_FLAG_AWD2_SLV

ADC flag ADC multimode slave analog watchdog 2 of the ADC slave

LL_ADC_FLAG_AWD3_MST

ADC flag ADC multimode master analog watchdog 3 of the ADC master

LL_ADC_FLAG_AWD3_SLV

ADC flag ADC multimode slave analog watchdog 3 of the ADC slave

ADC instance - Groups

LL_ADC_GROUP_REGULAR

ADC group regular (available on all STM32 devices)

LL_ADC_GROUP_INJECTED

ADC group injected (not available on all STM32 devices)

LL_ADC_GROUP_REGULAR_INJECTED

ADC both groups regular and injected

Definitions of ADC hardware constraints delays

LL_ADC_DELAY_INTERNAL_REGUL_STAB_US

Delay for ADC stabilization time (ADC voltage regulator start-up time)

LL_ADC_DELAY_VREFINT_STAB_US

Delay for internal voltage reference stabilization time

LL_ADC_DELAY_TEMPSENSOR_STAB_US

Delay for temperature sensor stabilization time

LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES

Delay required between ADC end of calibration and ADC enable

ADC_LINEARITY_BIT_TOGGLE_TIMEOUT

ADC linearity calibration set/clear bit delay

ADC group injected - Sequencer discontinuous mode

LL_ADC_INJ_SEQ_DISCONT_DISABLE

ADC group injected sequencer discontinuous mode disable

LL_ADC_INJ_SEQ_DISCONT_1RANK

ADC group injected sequencer discontinuous mode enable with sequence interruption every rank

ADC group injected - Sequencer ranks

LL_ADC_INJ_RANK_1

ADC group injected sequencer rank 1

LL_ADC_INJ_RANK_2

ADC group injected sequencer rank 2

LL_ADC_INJ_RANK_3

ADC group injected sequencer rank 3

LL_ADC_INJ_RANK_4

ADC group injected sequencer rank 4

ADC group injected - Sequencer scan length

LL_ADC_INJ_SEQ_SCAN_DISABLE

ADC group injected sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_INJ_SEQ_SCAN_ENABLE_2RANKS

ADC group injected sequencer enable with 2 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_3RANKS

ADC group injected sequencer enable with 3 ranks in the sequence

LL_ADC_INJ_SEQ_SCAN_ENABLE_4RANKS

ADC group injected sequencer enable with 4 ranks in the sequence

ADC group injected - Trigger edge

LL_ADC_INJ_TRIG_EXT_RISING

ADC group injected conversion trigger polarity set to rising edge

LL_ADC_INJ_TRIG_EXT_FALLING

ADC group injected conversion trigger polarity set to falling edge

LL_ADC_INJ_TRIG_EXT_RISINGFALLING

ADC group injected conversion trigger polarity set to both rising and falling edges

ADC group injected - Trigger source

LL_ADC_INJ_TRIG_SOFTWARE

ADC group injected conversion trigger internal: SW start.

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO

ADC group injected conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_CH4

ADC group injected conversion trigger from external peripheral: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_TRGO

ADC group injected conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM2_CH1

ADC group injected conversion trigger from external peripheral: TIM2 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH4

ADC group injected conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM4_TRGO

ADC group injected conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_EXTI_LINE15

ADC group injected conversion trigger from external peripheral: external interrupt line 15. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_CH4

ADC group injected conversion trigger from external peripheral: TIM8 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM1_TRGO2

ADC group injected conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO

ADC group injected conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM8_TRGO2

ADC group injected conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH3

ADC group injected conversion trigger from external peripheral: TIM3 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_TRGO

ADC group injected conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM3_CH1

ADC group injected conversion trigger from external peripheral: TIM3 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM6_TRGO

ADC group injected conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_TIM15_TRGO

ADC group injected conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM1_CH2

ADC group injected conversion trigger from external peripheral: LPTIM1 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM2_CH2

ADC group injected conversion trigger from external peripheral: LPTIM2 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM3_CH1

ADC group injected conversion trigger from external peripheral: LPTIM3 OUT event. 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_INJ_TRIG_EXT_LPTIM4_OUT

ADC group injected conversion trigger from external peripheral: LPTIM3 OUT event. 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

ADC group injected - Automatic trigger mode

LL_ADC_INJ_TRIG_INDEPENDENT

ADC group injected conversion trigger independent. Setting mandatory if ADC group injected injected trigger source is set to an external trigger.

LL_ADC_INJ_TRIG_FROM_GRP_REGULAR

ADC group injected conversion trigger from ADC group regular. Setting compliant only with group injected trigger source set to SW start, without any further action on ADC group injected conversion start or stop: in this case, ADC group injected is controlled only from ADC group regular.

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY

ADC interruption ADC instance ready

LL_ADC_IT_EOC

ADC interruption ADC group regular end of unitary conversion

LL_ADC_IT_EOS

ADC interruption ADC group regular end of sequence conversions

LL_ADC_IT_OVR

ADC interruption ADC group regular overrun

LL_ADC_IT_EOSMP

ADC interruption ADC group regular end of sampling phase

LL_ADC_IT_JEOC

ADC interruption ADC group injected end of unitary conversion

LL_ADC_IT_JEOS

ADC interruption ADC group injected end of sequence conversions

LL_ADC_IT_AWD1

ADC interruption ADC analog watchdog 1

LL_ADC_IT_AWD2

ADC interruption ADC analog watchdog 2

LL_ADC_IT_AWD3

ADC interruption ADC analog watchdog 3

LL_ADC_IT_EOCAL

ADC interruption ADC end of calibration

LL_ADC_IT_LDORDY

ADC interruption ADC Voltage Regulator (LDO) Ready

ADC left Shift

LL_ADC_LEFT_BIT_SHIFT_NONE

ADC no bit shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_1

ADC 1 bit shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_2

ADC 2 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_3

ADC 3 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_4

ADC 4 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_5

ADC 5 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_6

ADC 6 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_7

ADC 7 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_8

ADC 8 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_9

ADC 9 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_10

ADC 10 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_11

ADC 11 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_12

ADC 12 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_13

ADC 13 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_14

ADC 14 bits shift left applied on the final ADC conversion data

LL_ADC_LEFT_BIT_SHIFT_15

ADC 15 bits shift left applied on the final ADC conversion data

ADC instance - Low power mode

LL_ADC_LP_MODE_NONE

No ADC low power mode activated

LL_ADC_LP_AUTOWAIT

ADC low power mode auto delay: Dynamic low power mode, ADC conversions are performed only when necessary (when previous ADC conversion data is read). See description with function

Multimode - DMA transfer

LL_ADC_MULTI_REG_DMA_EACH_ADC

ADC multimode group regular conversions are transferred by DMA: each ADC uses its own DMA channel, with its individual DMA transfer settings

LL_ADC_MULTI_REG_DMA_RES_32_10B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master). Setting for ADC resolution of 32 (16x2) down to 10 bits

LL_ADC_MULTI_REG_DMA_RES_8B

ADC multimode group regular conversions are transferred by DMA, one DMA channel for both ADC (DMA of ADC master). Setting for ADC resolution of 8 bits

Multimode - ADC master or slave

LL_ADC_MULTI_MASTER

In multimode, selection among several ADC instances: ADC master

LL_ADC_MULTI_SLAVE

In multimode, selection among several ADC instances: ADC slave

LL_ADC_MULTI_MASTER_SLAVE

In multimode, selection among several ADC instances: both ADC master and ADC slave

Multimode - Mode

LL_ADC_MULTI_INDEPENDENT

ADC dual mode disabled (ADC independent mode)

LL_ADC_MULTI_DUAL_REG_SIMULT

ADC dual mode enabled: group regular simultaneous

LL_ADC_MULTI_DUAL_REG_INTERL

ADC dual mode enabled: Combined group regular interleaved

LL_ADC_MULTI_DUAL_INJ_SIMULT

ADC dual mode enabled: group injected simultaneous

LL_ADC_MULTI_DUAL_INJ_ALTERN

ADC dual mode enabled: group injected alternate trigger. Works only with external triggers (not internal SW start)

LL_ADC_MULTI_DUAL_REG_SIM_INJ_SIM

ADC dual mode enabled: Combined group regular simultaneous + group injected simultaneous

LL_ADC_MULTI_DUAL_REG_SIM_INJ_ALT

ADC dual mode enabled: Combined group regular simultaneous + group injected alternate trigger

LL_ADC_MULTI_DUAL_REG_INT_INJ_SIM

ADC dual mode enabled: Combined group regular interleaved + group injected simultaneous

Multimode - Delay between two sampling phases

LL_ADC_MULTI_TWOSMP_DELAY_1CYCLE

ADC multimode delay between two sampling phases: 1 ADC clock cycle for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_2CYCLES

ADC multimode delay between two sampling phases: 2 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_3CYCLES

ADC multimode delay between two sampling phases: 3 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_4CYCLES

ADC multimode delay between two sampling phases: 4 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_5CYCLES

ADC multimode delay between two sampling phases: 5 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_6CYCLES

ADC multimode delay between two sampling phases: 6 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_7CYCLES

ADC multimode delay between two sampling phases: 7 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_8CYCLES

ADC multimode delay between two sampling phases: 8 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_9CYCLES

ADC multimode delay between two sampling phases: 9 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_10CYCLES

ADC multimode delay between two sampling phases: 10 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_11CYCLES

ADC multimode delay between two sampling phases: 11 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_12CYCLES

ADC multimode delay between two sampling phases: 12 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES

ADC multimode delay between two sampling phases: 13 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_13CYCLES_8_BITS

ADC multimode delay between two sampling phases: 13 ADC clock cycles for 8 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_14CYCLES

ADC multimode delay between two sampling phases: 14 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES

ADC multimode delay between two sampling phases: 15 ADC clock cycles for all resolution

LL_ADC_MULTI_TWOSMP_DELAY_15CYCLES_10_BITS

ADC multimode delay between two sampling phases: 15 ADC clock cycles for 10 bits resolution

LL_ADC_MULTI_TWOSMP_DELAY_16CYCLES

ADC multimode delay between two sampling phases: 16 ADC clock cycles for all resolution

ADC instance - Offset number

LL_ADC_OFFSET_1

ADC offset number 1: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_2

ADC offset number 2: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_3

ADC offset number 3: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

LL_ADC_OFFSET_4

ADC offset number 4: ADC channel and offset level to which the offset programmed will be applied (independently of channel mapped on ADC group regular or group injected)

ADC instance - Offset sign

LL_ADC_OFFSET_SIGN_NEGATIVE

ADC offset is negative (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_SIGN_POSITIVE

ADC offset is positive (among ADC selected offset number 1, 2, 3 or 4)

ADC instance - Offset signed saturation mode

LL_ADC_OFFSET_SIGNED_SATURATION_DISABLE

ADC offset signed saturation is disabled (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_SIGNED_SATURATION_ENABLE

ADC offset signed saturation is enabled (among ADC selected offset number 1, 2, 3 or 4)

ADC instance - Offset unsigned saturation mode

LL_ADC_OFFSET_UNSIGNED_SATURATION_DISABLE

ADC offset unsigned saturation is disabled (among ADC selected offset number 1, 2, 3 or 4)

LL_ADC_OFFSET_UNSIGNED_SATURATION_ENABLE

ADC offset unsigned saturation is enabled (among ADC selected offset number 1, 2, 3 or 4)

Oversampling - Discontinuous mode

LL_ADC_OVS_REG_CONT

ADC oversampling discontinuous mode: continuous mode (all conversions of oversampling ratio are done from 1 trigger)

LL_ADC_OVS_REG_DISCONT

ADC oversampling discontinuous mode: discontinuous mode (each conversion of oversampling ratio needs a trigger)

Oversampling - Ratio

LL_ADC_OVS_RATIO_2

ADC oversampling ratio of 2 (2 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_4

ADC oversampling ratio of 4 (4 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_8

ADC oversampling ratio of 8 (8 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_16

ADC oversampling ratio of 16 (16 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_32

ADC oversampling ratio of 32 (32 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_64

ADC oversampling ratio of 64 (64 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_128

ADC oversampling ratio of 128 (128 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_256

ADC oversampling ratio of 256 (256 ADC conversions are performed, sum of these conversions data is computed to result as the ADC oversampling conversion data (before potential shift)

Oversampling - Oversampling scope

LL_ADC_OVS_DISABLE

ADC oversampling disabled.

LL_ADC_OVS_GRP_REGULAR_CONTINUED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is temporary stopped and continued afterwards.

LL_ADC_OVS_GRP_REGULAR_RESUMED

ADC oversampling on conversions of ADC group regular. If group injected interrupts group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

LL_ADC_OVS_GRP_INJECTED

ADC oversampling on conversions of ADC group injected.

LL_ADC_OVS_GRP_INJ_REG_RESUMED

ADC oversampling on conversions of both ADC groups regular and injected. If group injected interrupting group regular: when ADC group injected is triggered, the oversampling on ADC group regular is resumed from start (oversampler buffer reset).

Oversampling - Data shift

LL_ADC_OVS_SHIFT_NONE

ADC oversampling no shift (sum of the ADC conversions data is not divided to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_1

ADC oversampling shift of 1 (sum of the ADC conversions data is divided by 2 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_2

ADC oversampling shift of 2 (sum of the ADC conversions data is divided by 4 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_3

ADC oversampling shift of 3 (sum of the ADC conversions data is divided by 8 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_4

ADC oversampling shift of 4 (sum of the ADC conversions data is divided by 16 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_5

ADC oversampling shift of 5 (sum of the ADC conversions data is divided by 32 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_6

ADC oversampling shift of 6 (sum of the ADC conversions data is divided by 64 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_7

ADC oversampling shift of 7 (sum of the ADC conversions data is divided by 128 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_8

ADC oversampling shift of 8 (sum of the ADC conversions data is divided by 256 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_9

ADC oversampling shift of 9 (sum of the ADC conversions data is divided by 512 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_10

ADC oversampling shift of 10 (sum of the ADC conversions data is divided by 1024 to result as the ADC oversampling conversion data)

LL_ADC_OVS_SHIFT_RIGHT_11

ADC oversampling shift of 11 (sum of the ADC conversions data is divided by 2048 to result as the ADC oversampling conversion data)

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

LL_ADC_DMA_REG_REGULAR_DATA_MULTI

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE

ADC conversions are performed in single mode: one conversion per trigger

LL_ADC_REG_CONV_CONTINUOUS

ADC conversions are performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - Data transfer mode of ADC conversion data

LL_ADC_REG_DR_TRANSFER

ADC conversions are transferred to DR rigister

LL_ADC_REG_DMA_TRANSFER_NONE

ADC conversion data are not transferred by DMA but to DR register

LL_ADC_REG_DMA_TRANSFER_LIMITED

ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRANSFER_UNLIMITED

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

LL_ADC_REG_MDF_TRANSFER

ADC conversion data are transferred to DFSDM

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE_ADC4

ADC conversions are not transferred by DMA

LL_ADC_REG_DMA_TRANSFER_LIMITED_ADC4

ADC conversion data are transferred by DMA, in limited mode (one shot mode): DMA transfer requests are stopped when number of DMA data transfers (number of ADC conversions) is reached. This ADC mode is intended to be used with DMA mode non-circular.

LL_ADC_REG_DMA_TRANSFER_UNLIMITED_ADC4

ADC conversion data are transferred by DMA, in unlimited mode: DMA transfer requests are unlimited, whatever number of DMA data transferred (number of ADC conversions). This ADC mode is intended to be used with DMA mode circular.

ADC group regular - Overrun behavior on conversion data

LL_ADC_REG_OVR_DATA_PRESERVED

ADC group regular behavior in case of overrun: data preserved

LL_ADC_REG_OVR_DATA_OVERWRITTEN

ADC group regular behavior in case of overrun: data overwritten

ADC group regular - Sampling mode

LL_ADC_REG_SAMPLING_MODE_NORMAL

ADC conversions sampling phase duration is defined using

LL_ADC_REG_SAMPLING_MODE_BULB

ADC conversions sampling phase starts immediately after end of conversion, and stops upon trigger event. Note: First conversion is using minimal sampling time (see

LL_ADC_REG_SAMPLING_MODE_TRIGGER_CONTROLED

ADC conversions sampling phase is controlled by trigger events: Trigger rising edge = start sampling Trigger falling edge = stop sampling and start conversion.

ADC group regular - Sequencer discontinuous mode

LL_ADC_REG_SEQ_DISCONT_DISABLE

ADC group regular sequencer discontinuous mode disable

LL_ADC_REG_SEQ_DISCONT_1RANK

ADC group regular sequencer discontinuous mode enable with sequence interruption every rank

LL_ADC_REG_SEQ_DISCONT_2RANKS

ADC group regular sequencer discontinuous mode enabled with sequence interruption every 2 ranks

LL_ADC_REG_SEQ_DISCONT_3RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 3 ranks

LL_ADC_REG_SEQ_DISCONT_4RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 4 ranks

LL_ADC_REG_SEQ_DISCONT_5RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 5 ranks

LL_ADC_REG_SEQ_DISCONT_6RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 6 ranks

LL_ADC_REG_SEQ_DISCONT_7RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 7 ranks

LL_ADC_REG_SEQ_DISCONT_8RANKS

ADC group regular sequencer discontinuous mode enable with sequence interruption every 8 ranks

ADC group regular - Sequencer configuration flexibility

LL_ADC_REG_SEQ_FIXED

Sequencer configured to not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number. Refer to description of function

LL_ADC_REG_SEQ_CONFIGURABLE

Sequencer configured to fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function

ADC group regular - Sequencer ranks

LL_ADC_REG_RANK_1

ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2

ADC group regular sequencer rank 2

LL_ADC_REG_RANK_3

ADC group regular sequencer rank 3

LL_ADC_REG_RANK_4

ADC group regular sequencer rank 4

LL_ADC_REG_RANK_5

ADC group regular sequencer rank 5

LL_ADC_REG_RANK_6

ADC group regular sequencer rank 6

LL_ADC_REG_RANK_7

ADC group regular sequencer rank 7

LL_ADC_REG_RANK_8

ADC group regular sequencer rank 8

LL_ADC_REG_RANK_9

ADC group regular sequencer rank 9

LL_ADC_REG_RANK_10

ADC group regular sequencer rank 10

LL_ADC_REG_RANK_11

ADC group regular sequencer rank 11

LL_ADC_REG_RANK_12

ADC group regular sequencer rank 12

LL_ADC_REG_RANK_13

ADC group regular sequencer rank 13

LL_ADC_REG_RANK_14

ADC group regular sequencer rank 14

LL_ADC_REG_RANK_15

ADC group regular sequencer rank 15

LL_ADC_REG_RANK_16

ADC group regular sequencer rank 16

LL_ADC_REG_RANK_1_ADC4

ADC group regular sequencer rank 1

LL_ADC_REG_RANK_2_ADC4

ADC group regular sequencer rank 2

LL_ADC_REG_RANK_3_ADC4

ADC group regular sequencer rank 3

LL_ADC_REG_RANK_4_ADC4

ADC group regular sequencer rank 4

LL_ADC_REG_RANK_5_ADC4

ADC group regular sequencer rank 5

LL_ADC_REG_RANK_6_ADC4

ADC group regular sequencer rank 6

LL_ADC_REG_RANK_7_ADC4

ADC group regular sequencer rank 7

LL_ADC_REG_RANK_8_ADC4

ADC group regular sequencer rank 8

ADC group regular - Sequencer scan direction

LL_ADC_REG_SEQ_SCAN_DIR_FORWARD

On this STM32 series, parameter relevant only is sequencer set to mode not fully configurable, refer to function

LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD

On this STM32 series, parameter relevant only is sequencer set to mode not fully configurable, refer to function

ADC group regular - Sequencer scan length

LL_ADC_REG_SEQ_SCAN_DISABLE

ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC_REG_SEQ_SCAN_ENABLE_2RANKS

ADC group regular sequencer enable with 2 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS

ADC group regular sequencer enable with 3 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_4RANKS

ADC group regular sequencer enable with 4 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_5RANKS

ADC group regular sequencer enable with 5 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS

ADC group regular sequencer enable with 6 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_7RANKS

ADC group regular sequencer enable with 7 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_8RANKS

ADC group regular sequencer enable with 8 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_9RANKS

ADC group regular sequencer enable with 9 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_10RANKS

ADC group regular sequencer enable with 10 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_11RANKS

ADC group regular sequencer enable with 11 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_12RANKS

ADC group regular sequencer enable with 12 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_13RANKS

ADC group regular sequencer enable with 13 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_14RANKS

ADC group regular sequencer enable with 14 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_15RANKS

ADC group regular sequencer enable with 15 ranks in the sequence

LL_ADC_REG_SEQ_SCAN_ENABLE_16RANKS

ADC group regular sequencer enable with 16 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_DISABLE

ADC group regular sequencer disable (equivalent to sequencer of 1 rank: ADC conversion on only 1 channel)

LL_ADC4_REG_SEQ_SCAN_ENABLE_2RANKS

ADC group regular sequencer enable with 2 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_ENABLE_3RANKS

ADC group regular sequencer enable with 3 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_ENABLE_4RANKS

ADC group regular sequencer enable with 4 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_ENABLE_5RANKS

ADC group regular sequencer enable with 5 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_ENABLE_6RANKS

ADC group regular sequencer enable with 6 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_ENABLE_7RANKS

ADC group regular sequencer enable with 7 ranks in the sequence

LL_ADC4_REG_SEQ_SCAN_ENABLE_8RANKS

ADC group regular sequencer enable with 8 ranks in the sequence

ADC group regular - Trigger edge

LL_ADC_REG_TRIG_EXT_RISING

ADC group regular conversion trigger polarity set to rising edge

LL_ADC_REG_TRIG_EXT_FALLING

ADC group regular conversion trigger polarity set to falling edge

LL_ADC_REG_TRIG_EXT_RISINGFALLING

ADC group regular conversion trigger polarity set to both rising and falling edges

ADC group regular - Trigger frequency mode

LL_ADC_TRIGGER_FREQ_HIGH

ADC trigger frequency mode set to high frequency. Note: ADC trigger frequency mode must be set to low frequency when a duration is exceeded before ADC conversion start trigger event (between ADC enable and ADC conversion start trigger event or between two ADC conversion start trigger event). Duration value: Refer to device datasheet, parameter "tIdle".

LL_ADC_TRIGGER_FREQ_LOW

ADC trigger frequency mode set to low frequency. Note: ADC trigger frequency mode must be set to low frequency when a duration is exceeded before ADC conversion start trigger event (between ADC enable and ADC conversion start trigger event or between two ADC conversion start trigger event). Duration value: Refer to device datasheet, parameter "tIdle".

ADC group regular - Trigger source

LL_ADC_REG_TRIG_SOFTWARE

ADC group regular conversion trigger internal: SW start.

LL_ADC_REG_TRIG_EXT_TIM1_CH1

ADC group regular conversion trigger from external peripheral: TIM1 channel 1 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH2

ADC group regular conversion trigger from external peripheral: TIM1 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH3

ADC group regular conversion trigger from external peripheral: TIM1 channel 3 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_CH2

ADC group regular conversion trigger from external peripheral: TIM2 channel 2 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_TRGO

ADC group regular conversion trigger from external peripheral: TIM3 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_CH4

ADC group regular conversion trigger from external peripheral: TIM4 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_EXTI_LINE11

ADC group regular conversion trigger from external peripheral: external interrupt line 11 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO

ADC group regular conversion trigger from external peripheral: TIM8 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM8_TRGO2

ADC group regular conversion trigger from external peripheral: TIM8 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO

ADC group regular conversion trigger from external peripheral: TIM1 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO2

ADC group regular conversion trigger from external peripheral: TIM1 TRGO2 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO

ADC group regular conversion trigger from external peripheral: TIM2 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM4_TRGO

ADC group regular conversion trigger from external peripheral: TIM4 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM6_TRGO

ADC group regular conversion trigger from external peripheral: TIM6 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM15_TRGO

ADC group regular conversion trigger from external peripheral: TIM15 TRGO event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM3_CH4

ADC group regular conversion trigger from external peripheral: TIM3 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_EXTI_LINE15

ADC group regular conversion trigger from external peripheral: external interrupt line 15 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM1_CH1

ADC group regular conversion trigger from external peripheral: LPTIM1 channel 1 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM2_CH1

ADC group regular conversion trigger from external peripheral: LPTIM2 channel 1 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM3_CH1

ADC group regular conversion trigger from external peripheral: LPTIM3 channel 1 event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM4_OUT

ADC group regular conversion trigger from external peripheral: LPTIM4 OUT event. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_TRGO2_ADC4

ADC group regular conversion trigger from external IP: TIM1 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM1_CH4_ADC4

ADC group regular conversion trigger from external IP: TIM1 channel 4 event (capture compare: input capture or output capture). Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO_ADC4

ADC group regular conversion trigger from external IP: TIM2 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM15_TRGO_ADC4

ADC group regular conversion trigger from external IP: TIM15 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_TIM6_TRGO_ADC4

ADC group regular conversion trigger from external IP: TIM6 TRGO. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM1_CH1_ADC4

ADC group regular conversion trigger from external IP: LPTIM1 CH1. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_LPTIM3_CH2_ADC4

ADC group regular conversion trigger from external IP: LPTIM3 CH2. Trigger edge set to rising edge (default setting).

LL_ADC_REG_TRIG_EXT_EXTI_LINE15_ADC4

ADC group regular conversion trigger from external IP: external interrupt line 15. Trigger edge set to rising edge (default setting).

ADC instance - Resolution

LL_ADC_RESOLUTION_14B

ADC resolution 14 bits (ADC1, ADC2 only)

LL_ADC_RESOLUTION_12B

ADC resolution 12 bits

LL_ADC_RESOLUTION_10B

ADC resolution 10 bits

LL_ADC_RESOLUTION_8B

ADC resolution 8 bits

LL_ADC_RESOLUTION_6B

ADC resolution 6 bits (ADC4 only) (value shift out of ADC_CFGR1_RES range, post-processing when applied with ADC4)

LL_ADC_RESOLUTION_12B_ADC4

LL_ADC_RESOLUTION_10B_ADC4

LL_ADC_RESOLUTION_8B_ADC4

LL_ADC_RESOLUTION_6B_ADC4

ADC instance - Sampling time common to a group of channels

LL_ADC_SAMPLINGTIME_COMMON_1

Set sampling time common to a group of channels: sampling time nb 1

LL_ADC_SAMPLINGTIME_COMMON_2

Set sampling time common to a group of channels: sampling time nb 2

ADC4_SAMPLING_TIME_SMP_MASK

ADC4_SAMPLING_TIME_SMP_SHIFT_MASK

ADC instance - VREF protection mode

LL_ADC_VREF_PROT_DISABLE

ADC Vref+ protection disabled

LL_ADC_VREF_PROT_FIRST_SAMP_ENABLE

ADC Vref+ protection enabled: In case of simultaneous sampling phase of ADC4 and ADC1/2, ADC4 is put on hold during one ADC4 clock cycle to avoid noise on Vref+.

LL_ADC_VREF_PROT_SECOND_SAMP_ENABLE

ADC Vref+ protection enabled: In case of simultaneous sampling phase of ADC4 and ADC1/2, ADC4 is put on hold during two ADC4 clock cycles to avoid noise on Vref+.

ADC helper macro

__LL_ADC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get ADC channel number in decimal format from literals LL_ADC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  – LL_ADC_CHANNEL_0 (3)
  – LL_ADC_CHANNEL_1 (3)
  – LL_ADC_CHANNEL_2 (3)
  – LL_ADC_CHANNEL_3 (3)
  – LL_ADC_CHANNEL_4 (3)
  – LL_ADC_CHANNEL_5 (3)
  – LL_ADC_CHANNEL_6
  – LL_ADC_CHANNEL_7
  – LL_ADC_CHANNEL_8
  – LL_ADC_CHANNEL_9
  – LL_ADC_CHANNEL_10
  – LL_ADC_CHANNEL_11
  – LL_ADC_CHANNEL_12
  – LL_ADC_CHANNEL_13
  – LL_ADC_CHANNEL_14
  – LL_ADC_CHANNEL_15
  – LL_ADC_CHANNEL_16
  – LL_ADC_CHANNEL_17
  – LL_ADC_CHANNEL_18
  – LL_ADC_CHANNEL_19
  – LL_ADC_CHANNEL_VREFINT
  – LL_ADC_CHANNEL_TEMPSENSOR (1)
  – LL_ADC_CHANNEL_VBAT (1)
  – LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  – LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  – LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  – LL_ADC_CHANNEL_VBAT_ADC4 (2)
  – LL_ADC_CHANNEL_VCORE (2)

Return value:

- Value: between Min_Data=0 and Max_Data=18

Notes:

- Example: __LL_ADC_CHANNEL_TO_DECIMAL_NB(LL_ADC_CHANNEL_4) will return decimal number "4". The input can be a value from functions where a channel number is returned, either defined with number or with bitfield (only one bit must be set).

__LL_ADC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get ADC channel in literal format LL_ADC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: Value between Min_Data=0 and Max_Data=18

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)
  - LL_ADC_CHANNEL_VCORE (2)

Notes:

- Example: __LL_ADC_DECIMAL_NB_TO_CHANNEL(4) will return a data equivalent to "LL_ADC_CHANNEL_4".

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

- Helper macro to determine whether the selected channel corresponds to literal definitions of driver.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)
  - LL_ADC_CHANNEL_VCORE (2)

Return value:

- Value: "0" if the channel corresponds to a parameter definition of a ADC external channel (channel connected to a GPIO pin). Value "1" if the channel corresponds to a parameter definition of a ADC internal channel.

Notes:

- The different literal definitions of ADC channels are: ADC internal channel: LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...ADC external channel (channel connected to a GPIO pin): LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ... The channel parameter must be a value defined from literal definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...), must not be a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_CHANNEL_INTERNAL_TO_EXTERNAL

Description:

- Helper macro to convert a channel defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), to its equivalent parameter definition of a ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...).

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)
  - LL_ADC_CHANNEL_VCORE (2)

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_CHANNEL_0
  - LL_ADC_CHANNEL_1
  - LL_ADC_CHANNEL_2
  - LL_ADC_CHANNEL_3
  - LL_ADC_CHANNEL_4
  - LL_ADC_CHANNEL_5
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19

Notes:

- The channel parameter can be, additionally to a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers.

__LL_ADC_IS_CHANNEL_INTERNAL_AVAILABLE

Description:

- Helper macro to determine whether the internal channel selected is available on the ADC instance selected.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)
  - LL_ADC_CHANNEL_VCORE (2)

Return value:

- Value: "0" if the internal channel selected is not available on the ADC instance selected. Value "1" if the internal channel selected is available on the ADC instance selected.

Notes:

- The channel parameter must be a value defined from parameter definition of a ADC internal channel (LL_ADC_CHANNEL_VREFINT, LL_ADC_CHANNEL_TEMPSENSOR, ...), must not be a value defined from parameter definition of ADC external channel (LL_ADC_CHANNEL_1, LL_ADC_CHANNEL_2, ...) or a value from functions where a channel number is returned from ADC registers, because internal and external channels share the same channel number in ADC registers. The differentiation is made only with parameters definitions of driver.

__LL_ADC_ANALOGWD_CHANNEL_GROUP

Description:

- Helper macro to define ADC analog watchdog parameter: define a single channel to monitor with analog watchdog from sequencer channel and groups definition.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_0 (3)
  - LL_ADC_CHANNEL_1 (3)
  - LL_ADC_CHANNEL_2 (3)
  - LL_ADC_CHANNEL_3 (3)
  - LL_ADC_CHANNEL_4 (3)
  - LL_ADC_CHANNEL_5 (3)
  - LL_ADC_CHANNEL_6
  - LL_ADC_CHANNEL_7
  - LL_ADC_CHANNEL_8
  - LL_ADC_CHANNEL_9
  - LL_ADC_CHANNEL_10
  - LL_ADC_CHANNEL_11
  - LL_ADC_CHANNEL_12
  - LL_ADC_CHANNEL_13
  - LL_ADC_CHANNEL_14
  - LL_ADC_CHANNEL_15
  - LL_ADC_CHANNEL_16
  - LL_ADC_CHANNEL_17
  - LL_ADC_CHANNEL_18
  - LL_ADC_CHANNEL_19
  - LL_ADC_CHANNEL_VREFINT
  - LL_ADC_CHANNEL_TEMPSENSOR (1)
  - LL_ADC_CHANNEL_VBAT (1)
  - LL_ADC_CHANNEL_DAC1CH1_ADC4 (2)
  - LL_ADC_CHANNEL_DAC1CH2_ADC4 (2)
  - LL_ADC_CHANNEL_TEMPSENSOR_ADC4 (2)
  - LL_ADC_CHANNEL_VBAT_ADC4 (2)
  - LL_ADC_CHANNEL_VCORE (2)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR
  - LL_ADC_GROUP_INJECTED
  - LL_ADC_GROUP_REGULAR_INJECTED

Return value:

- Returned: value can be one of the following values:
  – LL_ADC_AWD_DISABLE
  – LL_ADC_AWD_ALL_CHANNELS_REG (0)
  – LL_ADC_AWD_ALL_CHANNELS_INJ (0)
  – LL_ADC_AWD_ALL_CHANNELS_REG_INJ
  – LL_ADC_AWD_CHANNEL_0_REG (0)
  – LL_ADC_AWD_CHANNEL_0_INJ (0)
  – LL_ADC_AWD_CHANNEL_0_REG_INJ
  – LL_ADC_AWD_CHANNEL_1_REG (0)
  – LL_ADC_AWD_CHANNEL_1_INJ (0)
  – LL_ADC_AWD_CHANNEL_1_REG_INJ
  – LL_ADC_AWD_CHANNEL_2_REG (0)
  – LL_ADC_AWD_CHANNEL_2_INJ (0)
  – LL_ADC_AWD_CHANNEL_2_REG_INJ
  – LL_ADC_AWD_CHANNEL_3_REG (0)
  – LL_ADC_AWD_CHANNEL_3_INJ (0)
  – LL_ADC_AWD_CHANNEL_3_REG_INJ
  – LL_ADC_AWD_CHANNEL_4_REG (0)
  – LL_ADC_AWD_CHANNEL_4_INJ (0)
  – LL_ADC_AWD_CHANNEL_4_REG_INJ
  – LL_ADC_AWD_CHANNEL_5_REG (0)
  – LL_ADC_AWD_CHANNEL_5_INJ (0)
  – LL_ADC_AWD_CHANNEL_5_REG_INJ
  – LL_ADC_AWD_CHANNEL_6_REG (0)
  – LL_ADC_AWD_CHANNEL_6_INJ (0)
  – LL_ADC_AWD_CHANNEL_6_REG_INJ
  – LL_ADC_AWD_CHANNEL_7_REG (0)
  – LL_ADC_AWD_CHANNEL_7_INJ (0)
  – LL_ADC_AWD_CHANNEL_7_REG_INJ
  – LL_ADC_AWD_CHANNEL_8_REG (0)
  – LL_ADC_AWD_CHANNEL_8_INJ (0)
  – LL_ADC_AWD_CHANNEL_8_REG_INJ
  – LL_ADC_AWD_CHANNEL_9_REG (0)
  – LL_ADC_AWD_CHANNEL_9_INJ (0)
  – LL_ADC_AWD_CHANNEL_9_REG_INJ
  – LL_ADC_AWD_CHANNEL_10_REG (0)
  – LL_ADC_AWD_CHANNEL_10_INJ (0)
  – LL_ADC_AWD_CHANNEL_10_REG_INJ
  – LL_ADC_AWD_CHANNEL_11_REG (0)
  – LL_ADC_AWD_CHANNEL_11_INJ (0)
  – LL_ADC_AWD_CHANNEL_11_REG_INJ
  – LL_ADC_AWD_CHANNEL_12_REG (0)
  – LL_ADC_AWD_CHANNEL_12_INJ (0)
  – LL_ADC_AWD_CHANNEL_12_REG_INJ
  – LL_ADC_AWD_CHANNEL_13_REG (0)
  – LL_ADC_AWD_CHANNEL_13_INJ (0)
  – LL_ADC_AWD_CHANNEL_13_REG_INJ
  – LL_ADC_AWD_CHANNEL_14_REG (0)
  – LL_ADC_AWD_CHANNEL_14_INJ (0)

- LL_ADC_AWD_CHANNEL_14_REG_INJ
- LL_ADC_AWD_CHANNEL_15_REG (0)
- LL_ADC_AWD_CHANNEL_15_INJ (0)
- LL_ADC_AWD_CHANNEL_15_REG_INJ
- LL_ADC_AWD_CHANNEL_16_REG (0)
- LL_ADC_AWD_CHANNEL_16_INJ (0)
- LL_ADC_AWD_CHANNEL_16_REG_INJ
- LL_ADC_AWD_CHANNEL_17_REG (0)
- LL_ADC_AWD_CHANNEL_17_INJ (0)
- LL_ADC_AWD_CHANNEL_17_REG_INJ
- LL_ADC_AWD_CHANNEL_18_REG (0)
- LL_ADC_AWD_CHANNEL_18_INJ (0)
- LL_ADC_AWD_CHANNEL_18_REG_INJ
- LL_ADC_AWD_CHANNEL_19_REG (0)
- LL_ADC_AWD_CHANNEL_19_INJ (0)
- LL_ADC_AWD_CHANNEL_19_REG_INJ
- LL_ADC_AWD_CH_VREFINT_REG (0)(1)
- LL_ADC_AWD_CH_VREFINT_INJ (0)(1)
- LL_ADC_AWD_CH_VREFINT_REG_INJ (1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG (0)(1)
- LL_ADC_AWD_CH_TEMPSENSOR_INJ (0)(1)
- LL_ADC_AWD_CH_TEMPSENSOR_REG_INJ (1)
- LL_ADC_AWD_CH_VBAT_REG (0)(1)
- LL_ADC_AWD_CH_VBAT_INJ (0)(1)
- LL_ADC_AWD_CH_VBAT_REG_INJ (1)
- LL_ADC_AWD_CH_DAC1CH1_ADC4_REG (0)(2)
- LL_ADC_AWD_CH_DAC1CH2_ADC4_REG (0)(2)
- LL_ADC_AWD_CH_TEMPSENSOR_ADC4_REG (2)
- LL_ADC_AWD_CH_VBAT_ADC4_REG (2)
- LL_ADC_AWD_CH_VCORE_REG (2)

Notes:

- To be used with function LL_ADC_SetAnalogWDMonitChannels(). Example: LL_ADC_SetAnalogWDMonitChannels(ADC1, LL_ADC_AWD1, __LL_ADC_ANALOGWD_CHANNEL_GROUP(LL_ADC_CHANNEL4, LL_ADC_GROUP_REGULAR))

__LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION

Description:

- Helper macro to set the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 14 bits in case of ADC1 or ADC2, 12 bits in case of ADC4.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  – LL_ADC_RESOLUTION_14B (1)
  – LL_ADC_RESOLUTION_12B
  – LL_ADC_RESOLUTION_10B
  – LL_ADC_RESOLUTION_8B
  – LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4
- __AWD_THRESHOLD__: Value between Min_Data=0x000000 and Max_Data=0x1FFFFFF, In case of ADC1 instance __AWD_THRESHOLD__ Value between Min_Data=0x000 and Max_Data=0xFFF, In case of ADC4 instance

Return value:

- In: case of ADC1 instance, Value between Min_Data=0x000000 and Max_Data=0x1FFFFFF In case of ADC4 instance, Value between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_SetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to set the value of analog watchdog threshold high (on 8 bits): LL_ADC_SetAnalogWDThresholds (< ADCx param >, __LL_ADC_ANALOGWD_SET_THRESHOLD_RESOLUTION(LL_ADC_RESOLUTION_8B, <threshold_value_8_bits>));

__LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION

Description:

- Helper macro to get the value of ADC analog watchdog threshold high or low in function of ADC resolution, when ADC resolution is different of 14 bits in case of ADC1 or ADC2, 12 bits in case of ADC4.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  – LL_ADC_RESOLUTION_14B (1)
  – LL_ADC_RESOLUTION_12B
  – LL_ADC_RESOLUTION_10B
  – LL_ADC_RESOLUTION_8B
  – LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4
- __AWD_THRESHOLD__: Value between Min_Data=0x000000 and Max_Data=0x1FFFFFF, In case of ADC1 instance __AWD_THRESHOLD__ Value between Min_Data=0x000 and Max_Data=0xFFF, In case of ADC4 instance

Return value:

- In: case of ADC1 instance, Value between Min_Data=0x000000 and Max_Data=0x1FFFFFF In case of ADC4 instance, Value between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, with a ADC resolution of 8 bits, to get the value of analog watchdog threshold high (on 8 bits): < threshold_value_8_bits > = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_CALIB_FACTOR_SINGLE_DIFF

Description:

- Helper macro to set the ADC calibration value with both single ended and differential modes calibration factors concatenated.

Parameters:

- __CALIB_FACTOR_SINGLE_ENDED__: Value between Min_Data=0x00 and Max_Data=0x7F
- __CALIB_FACTOR_DIFFERENTIAL__: Value between Min_Data=0x00 and Max_Data=0x7F

Return value:

- Value: between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Notes:

- To be used with function LL_ADC_SetCalibrationOffsetFactor(). Example, to set calibration factors single ended to 0x55 and differential ended to 0x2A: LL_ADC_SetCalibrationOffsetFactor(ADC1, __LL_ADC_CALIB_FACTOR_SINGLE_DIFF(0x55, 0x2A))

__LL_ADC_MULTI_CONV_DATA_MASTER_SLAVE

Description:

- Helper macro to get the ADC multimode conversion data of ADC master or ADC slave from raw value with both ADC conversion data concatenated.

Parameters:

- __ADC_MULTI_MASTER_SLAVE__: This parameter can be one of the following values:
  - LL_ADC_MULTI_MASTER
  - LL_ADC_MULTI_SLAVE
- __ADC_MULTI_CONV_DATA__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- Value: between Min_Data=0x000 and Max_Data=0xFFF

Notes:

- This macro is intended to be used when multimode transfer by DMA is enabled: refer to function LL_ADC_SetMultiDMATransfer(). In this case the transferred data need to processed with this macro to separate the conversion data of ADC master and ADC slave.

__LL_ADC_MULTI_INSTANCE_MASTER

Description:

- Helper macro to select, from a ADC instance, to which ADC instance it has a dependence in multimode (ADC master of the corresponding ADC common instance).

Parameters:

- __ADCx__: ADC instance

Return value:

- __ADCx__: ADC instance master of the corresponding ADC common instance

Notes:

- In case of device with multimode available and a mix of ADC instances compliant and not compliant with multimode feature, ADC instances not compliant with multimode feature are considered as master instances (do not depend to any other ADC instance).

__LL_ADC_COMMON_INSTANCE

Description:

- Helper macro to select the ADC common instance to which is belonging the selected ADC instance.

Parameters:

- __ADCx__: ADC instance

Return value:

- ADC: common register instance

Notes:

- ADC common register instance can be used for: Set parameters common to several ADC instancesMultimode (for devices with several ADC instances) Refer to functions having argument "ADCxy_COMMON" as parameter.

__LL_ADC_IS_ENABLED_ALL_COMMON_INSTANCE

Description:

- Helper macro to check if all ADC instances sharing the same ADC common instance are disabled.

Parameters:

- __ADCXY_COMMON__: ADC common instance (can be set directly from CMSIS definition or by using helper macro

Return value:

- Value: "0" if all ADC instances sharing the same ADC common instance are disabled. Value "1" if at least one ADC instance sharing the same ADC common instance is enabled.

Notes:

- This check is required by functions with setting conditioned to ADC state: All ADC instances of the ADC common group must be disabled. Refer to functions having argument "ADCxy_COMMON" as parameter. On devices with only 1 ADC common instance, parameter of this macro is useless and can be ignored (parameter kept for compatibility with devices featuring several ADC common instances).

__LL_ADC_DIGITAL_SCALE

Description:

- Helper macro to define the ADC conversion data full-scale digital value corresponding to the selected ADC resolution.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data full-scale digital value (unit: digital value of ADC conversion data)

Notes:

- ADC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_ADC_CONVERT_DATA_RESOLUTION

Description:

- Helper macro to convert the ADC conversion data from a resolution to another resolution.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __DATA__: ADC conversion data to be converted
- __ADC_RESOLUTION_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Return value:

- ADC: conversion data to the requested resolution

__LL_ADC_CALC_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (resolution 16 bits) (unit: digital value).
- __ADC_INSTANCE__: ADC instance
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B (1)
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_ADC_CALC_DIFF_DATA_TO_VOLTAGE

Description:

- Helper macro to calculate the voltage (unit: mVolt) corresponding to a ADC conversion data (unit: digital value) in differential ended mode.

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __ADC_DATA__: ADC conversion data (unit: digital value).
- __ADC_INSTANCE__: ADC instance
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- On STM32U5, this feature is available on ADC instances: ADC1, ADC2. ADC data from ADC data register is unsigned and centered around middle code in. Converted voltage can be positive or negative depending on differential input voltages. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

__LL_ADC_CALC_VREFANALOG_VOLTAGE

Description:

- Helper macro to calculate analog reference voltage (Vref+) (unit: mVolt) from ADC conversion data of internal voltage reference VrefInt.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __VREFINT_ADC_DATA__: ADC conversion data (resolution 16 bits) of internal voltage reference VrefInt (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B (1)
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4

Return value:

- Analog: reference voltage (unit: mV)

Notes:

- Computation is using VrefInt calibration value stored in system memory for each device during production. This voltage depends on user board environment: voltage level connected to pin Vref+. On devices with small package, the pin Vref+ is not present and internally bonded to pin Vdda. On this STM32 series, calibration data of internal voltage reference VrefInt corresponds to a resolution of 16 bits, this is the recommended ADC resolution to convert voltage of internal voltage reference VrefInt. Otherwise, this macro performs the processing to scale ADC conversion data to 14 bits.

__LL_ADC_CALC_TEMPERATURE

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B (1)
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = ((TS_ADC_DATA - TS_CAL1) * (TS_CAL2_TEMP - TS_CAL1_TEMP)) / (TS_CAL2 - TS_CAL1) + TS_CAL1_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC Avg_Slope = (TS_CAL2 - TS_CAL1) / (TS_CAL2_TEMP - TS_CAL1_TEMP) TS_CAL1 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) TS_CAL2 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). On this STM32 series, calibration data of temperature sensor corresponds to a resolution of 14 bits, this is the recommended ADC resolution to convert voltage of temperature sensor. Otherwise, this macro performs the processing to scale ADC conversion data to 14 bits.

__LL_ADC_CALC_TEMPERATURE_TYP_PARAMS

Description:

- Helper macro to calculate the temperature (unit: degree Celsius) from ADC conversion data of internal temperature sensor.

Parameters:

- __ADC_INSTANCE__: ADC instance
- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On STM32U5, refer to device datasheet parameter "Avg_Slope".
- __TEMPSENSOR_TYP_CALX_V__: Device datasheet data: Temperature sensor voltage typical value (at temperature and Vref+ defined in parameters below) (unit: mV). On STM32U5, refer to device datasheet parameter "V30" (corresponding to TS_CAL1).
- __TEMPSENSOR_CALX_TEMP__: Device datasheet data: Temperature at which temperature sensor voltage see parameter above) is corresponding (unit: mV)
- __VREFANALOG_VOLTAGE__: Analog voltage reference (Vref+) voltage (unit: mV)
- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_RESOLUTION__: ADC resolution at which internal temperature sensor voltage has been measured. This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_14B (1)
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B (2) (1): Specific to ADC instance: ADC1, ADC2 (2): Specific to ADC instance: ADC4

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (TS_TYP_CALx_VOLT(uV) - TS_ADC_DATA * Conversion_uV) / Avg_Slope + CALx_TEMP with TS_ADC_DATA = temperature sensor raw data measured by ADC (unit: digital value) Avg_Slope = temperature sensor slope (unit: uV/Degree Celsius) TS_TYP_CALx_VOLT = temperature sensor digital value at temperature CALx_TEMP (unit: mV) Caution: Calculation relevancy under reserve the temperature sensor of the current device has characteristics in line with datasheet typical values. If temperature sensor calibration values are available on on this device (presence of macro __LL_ADC_CALC_TEMPERATURE()), temperature calculation will be more accurate using helper macro __LL_ADC_CALC_TEMPERATURE(). As calculation input, the analog reference voltage (Vref+) must be defined as it impacts the ADC LSB equivalent voltage. Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE(). ADC measurement data must correspond to a resolution of 14 bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 14 bits.

Common write and read registers Macros

LL_ADC_WriteReg

Description:

- Write a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ADC_ReadReg

Description:

- Read a value in ADC register.

Parameters:

- __INSTANCE__: ADC Instance
- __REG__: Register to be read

Return value:

- Register: value

91 LL BUS Generic Driver

91.1 BUS Firmware driver API description

The following section lists the various functions of the BUS library.

91.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableBusClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableBusClock (void)

Function description

Enable AHB1 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB1DIS LL_AHB1_GRP1_EnableBusClock

LL_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_GPDMA1
  – LL_AHB1_GRP1_PERIPH_CORDIC
  – LL_AHB1_GRP1_PERIPH_FMAC
  – LL_AHB1_GRP1_PERIPH_MDF1
  – LL_AHB1_GRP1_PERIPH_FLASH
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_JPEG (*)
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_RAMCFG
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  – LL_AHB1_GRP1_PERIPH_GPU2D (*)
  – LL_AHB1_GRP1_PERIPH_DCACHE2 (*)
  – LL_AHB1_GRP1_PERIPH_GTZC1
  – LL_AHB1_GRP1_PERIPH_BKPSRAM
  – LL_AHB1_GRP1_PERIPH_DCACHE1
  – LL_AHB1_GRP1_PERIPH_SRAM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_EnableClock
- AHB1ENR FMACEN LL_AHB1_GRP1_EnableClock
- AHB1ENR MDF1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_EnableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_EnableClock
- AHB1ENR JPEGEN LL_AHB1_GRP1_EnableClock
- AHB1ENR TSCEN LL_AHB1_GRP1_EnableClock
- AHB1ENR RAMCFGEN LL_AHB1_GRP1_EnableClock
- AHB1ENR DMA2DEN LL_AHB1_GRP1_EnableClock
- AHB1ENR GFXMMUEN LL_AHB1_GRP1_EnableClock
- AHB1ENR GPU2DEN LL_AHB1_GRP1_EnableClock
- AHB1ENR DCACHE2EN LL_AHB1_GRP1_EnableClock
- AHB1ENR GTZC1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR BKPSRAMEN LL_AHB1_GRP1_EnableClock
- AHB1ENR DCACHE1EN LL_AHB1_GRP1_EnableClock
- AHB1ENR SRAM1EN LL_AHB1_GRP1_EnableClock
-

LL_AHB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_GPDMA1
  – LL_AHB1_GRP1_PERIPH_CORDIC
  – LL_AHB1_GRP1_PERIPH_FMAC
  – LL_AHB1_GRP1_PERIPH_MDF1
  – LL_AHB1_GRP1_PERIPH_FLASH
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_JPEG (*)
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_RAMCFG
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  – LL_AHB1_GRP1_PERIPH_GPU2D (*)
  – LL_AHB1_GRP1_PERIPH_DCACHE2 (*)
  – LL_AHB1_GRP1_PERIPH_GTZC1
  – LL_AHB1_GRP1_PERIPH_BKPSRAM
  – LL_AHB1_GRP1_PERIPH_DCACHE1
  – LL_AHB1_GRP1_PERIPH_SRAM1

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FMACEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR MDF1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR CRCEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR JPEGEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR TSCEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR RAMCFGEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DMA2DEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR GFXMMUEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR GPU2DEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DCACHE2EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR GTZC1EN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR BKPSRAMEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR DCACHEEN LL_AHB1_GRP1_IsEnabledClock
- AHB1ENR SRAM1EN LL_AHB1_GRP1_IsEnabledClock
-

LL_AHB1_GRP1_DisableBusClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableBusClock (void)

Function description

Disable AHB1 bus clock.

Return values

- **None:**

Notes

- except for FLASH, BKPSRAM, ICACHE, DCACHE1 and SRAM1.

Reference Manual to LL API cross reference:

- CFGR2 AHB1DIS LL_AHB1_GRP1_DisableBusClock

LL_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_GPDMA1
  – LL_AHB1_GRP1_PERIPH_CORDIC
  – LL_AHB1_GRP1_PERIPH_FMAC
  – LL_AHB1_GRP1_PERIPH_MDF1
  – LL_AHB1_GRP1_PERIPH_FLASH
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_JPEG (*)
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_RAMCFG
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  – LL_AHB1_GRP1_PERIPH_GPU2D (*)
  – LL_AHB1_GRP1_PERIPH_DCACHE2 (*)
  – LL_AHB1_GRP1_PERIPH_GTZC1
  – LL_AHB1_GRP1_PERIPH_BKPSRAM
  – LL_AHB1_GRP1_PERIPH_DCACHE1
  – LL_AHB1_GRP1_PERIPH_SRAM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1ENR GPDMA1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR CORDICEN LL_AHB1_GRP1_DisableClock
- AHB1ENR FMACEN LL_AHB1_GRP1_DisableClock
- AHB1ENR MDF1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR FLASHEN LL_AHB1_GRP1_DisableClock
- AHB1ENR CRCEN LL_AHB1_GRP1_DisableClock
- AHB1ENR JPEGEN LL_AHB1_GRP1_DisableClock
- AHB1ENR TSCEN LL_AHB1_GRP1_DisableClock
- AHB1ENR RAMCFGEN LL_AHB1_GRP1_DisableClock
- AHB1ENR DMA2DEN LL_AHB1_GRP1_DisableClock
- AHB1ENR GFXMMUEN LL_AHB1_GRP1_DisableClock
- AHB1ENR GPU2DEN LL_AHB1_GRP1_DisableClock
- AHB1ENR DCACHE2EN LL_AHB1_GRP1_DisableClock
- AHB1ENR GTZC1EN LL_AHB1_GRP1_DisableClock
- AHB1ENR BKPSRAMEN LL_AHB1_GRP1_DisableClock
- AHB1ENR DCACHEEN LL_AHB1_GRP1_DisableClock
- AHB1ENR SRAM1EN LL_AHB1_GRP1_DisableClock
-

LL_AHB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_GPDMA1
  – LL_AHB1_GRP1_PERIPH_CORDIC
  – LL_AHB1_GRP1_PERIPH_FMAC
  – LL_AHB1_GRP1_PERIPH_MDF1
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_JPEG (*)
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_RAMCFG
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  – LL_AHB1_GRP1_PERIPH_GPU2D (*)
  – LL_AHB1_GRP1_PERIPH_DCACHE2 (*) (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR GPDMA1RSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR CORDICRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR FMACRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR MDF1RSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR CRCRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR JPEGRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR TSCRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR RAMCFGRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR DMA2DRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR GFXMMURSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR GPU2DRSTR LL_AHB1_GRP1_ForceReset
- AHB1RSTR DCACHE2RSTR LL_AHB1_GRP1_ForceReset
-

LL_AHB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_MDF1
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_JPEG (*)
  - LL_AHB1_GRP1_PERIPH_TSC
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_DMA2D (*)
  - LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB1_GRP1_PERIPH_GPU2D (*)
  - LL_AHB1_GRP1_PERIPH_DCACHE2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1RSTR GPDMA1RSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CORDICRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR FMACRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR MDF1RSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR CRCRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR JPEGRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR TSCRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR RAMCFGRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DMA2DRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR GFXMMURSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR GPU2DRSTR LL_AHB1_GRP1_ReleaseReset
- AHB1RSTR DCACHE2RSTR LL_AHB1_GRP1_ReleaseReset
-

LL_AHB1_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_GPDMA1
  – LL_AHB1_GRP1_PERIPH_CORDIC
  – LL_AHB1_GRP1_PERIPH_FMAC
  – LL_AHB1_GRP1_PERIPH_MDF1
  – LL_AHB1_GRP1_PERIPH_FLASH
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_JPEG (*)
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_RAMCFG
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  – LL_AHB1_GRP1_PERIPH_GPU2D (*)
  – LL_AHB1_GRP1_PERIPH_DCACHE2 (*)
  – LL_AHB1_GRP1_PERIPH_GTZC1
  – LL_AHB1_GRP1_PERIPH_BKPSRAM
  – LL_AHB1_GRP1_PERIPH_ICACHE1
  – LL_AHB1_GRP1_PERIPH_DCACHE1
  – LL_AHB1_GRP1_PERIPH_SRAM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1SMENR GPDMA1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR CORDICSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR FMACSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR MDF1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR JPEGSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR TSCSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR RAMCFGSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DMA2DSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR GFXMMUSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR GPU2DSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DCACHE2SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR GTZC1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR BKPSRAMSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR ICACHESMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR DCACHESMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_EnableClockStopSleep
-

LL_AHB1_GRP1_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_AHB1_GRP1_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if AHB1 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_ALL
  – LL_AHB1_GRP1_PERIPH_GPDMA1
  – LL_AHB1_GRP1_PERIPH_CORDIC
  – LL_AHB1_GRP1_PERIPH_FMAC
  – LL_AHB1_GRP1_PERIPH_MDF1
  – LL_AHB1_GRP1_PERIPH_FLASH
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_JPEG (*)
  – LL_AHB1_GRP1_PERIPH_TSC
  – LL_AHB1_GRP1_PERIPH_RAMCFG
  – LL_AHB1_GRP1_PERIPH_DMA2D (*)
  – LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  – LL_AHB1_GRP1_PERIPH_GPU2D (*)
  – LL_AHB1_GRP1_PERIPH_DCACHE2 (*)
  – LL_AHB1_GRP1_PERIPH_GTZC1
  – LL_AHB1_GRP1_PERIPH_BKPSRAM
  – LL_AHB1_GRP1_PERIPH_ICACHE1
  – LL_AHB1_GRP1_PERIPH_DCACHE1
  – LL_AHB1_GRP1_PERIPH_SRAM1

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB1SMENR GPDMA1SMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR CORDICSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR FMACSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR MDF1SMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR JPEGSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR TSCSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR RAMCFGSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR DMA2DSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR GFXMMUSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR GPU2DSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR DCACHE2SMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR GTZC1SMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR BKPSRAMSMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR ICACHESMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR DCACHESMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_IsEnabledClockStopSleep
-

LL_AHB1_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_ALL
  - LL_AHB1_GRP1_PERIPH_GPDMA1
  - LL_AHB1_GRP1_PERIPH_CORDIC
  - LL_AHB1_GRP1_PERIPH_FMAC
  - LL_AHB1_GRP1_PERIPH_MDF1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_JPEG (*)
  - LL_AHB1_GRP1_PERIPH_TSC
  - LL_AHB1_GRP1_PERIPH_RAMCFG
  - LL_AHB1_GRP1_PERIPH_DMA2D (*)
  - LL_AHB1_GRP1_PERIPH_GFXMMU (*)
  - LL_AHB1_GRP1_PERIPH_GPU2D (*)
  - LL_AHB1_GRP1_PERIPH_DCACHE2 (*)
  - LL_AHB1_GRP1_PERIPH_GTZC1
  - LL_AHB1_GRP1_PERIPH_BKPSRAM
  - LL_AHB1_GRP1_PERIPH_ICACHE1
  - LL_AHB1_GRP1_PERIPH_DCACHE1
  - LL_AHB1_GRP1_PERIPH_SRAM1

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB1SMENR GPDMA1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR CORDICSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR FMACSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR MDF1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR FLASHSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR CRCSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR JPEGSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR TSCSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR RAMCFGSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DMA2DSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR GFXMMUSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR GPU2DSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DCACHE2SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR GTZC1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR BKPSRAMSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR ICACHESMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR DCACHESMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHB1SMENR SRAM1SMEN LL_AHB1_GRP1_DisableClockStopSleep
-

LL_AHB2_GRP1_EnableBusClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableBusClock (void)

Function description

Enable AHB2_1 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB2DIS1 LL_AHB2_GRP1_EnableBusClock

LL_AHB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  - LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  - LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  - LL_AHB2_GRP1_PERIPH_USBPHY (*)
  - LL_AHB2_GRP1_PERIPH_AES
  - LL_AHB2_GRP1_PERIPH_HASH
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB2_GRP1_PERIPH_SDMMC1
  - LL_AHB2_GRP1_PERIPH_SDMMC2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR1 GPIOAEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOBEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOCEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIODEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOEEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOFEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOHEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOIEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 GPIOJEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 ADC12EN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 DCMI_PSSIEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 OTGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 USBPHYCEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 AESEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 HASHEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 RNGEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 PKAEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 SAESEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 OCTOSPIMEN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 OTFDEC1EN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 OTFDEC2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 SDMMC1EN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 SDMMC2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 SRAM2EN LL_AHB2_GRP1_EnableClock
- AHB2ENR1 SRAM3EN LL_AHB2_GRP1_EnableClock
-

LL_AHB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  - LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  - LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  - LL_AHB2_GRP1_PERIPH_USBPHY (*)
  - LL_AHB2_GRP1_PERIPH_AES
  - LL_AHB2_GRP1_PERIPH_HASH
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB2_GRP1_PERIPH_SDMMC1
  - LL_AHB2_GRP1_PERIPH_SDMMC2 (*)
  - LL_AHB2_GRP1_PERIPH_SRAM2
  - LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2ENR1 GPIOAEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOBEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOCEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIODEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOEEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOFEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOHEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOIEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 GPIOJEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 ADC12EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 DCMI_PSSIEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 OTGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 USBPHYCEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 AESEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 HASHEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 RNGEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 PKAEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 SAESEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 OCTOSPIMEN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 OTFDEC1EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 OTFDEC2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 SDMMC1EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 SDMMC2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 SRAM2EN LL_AHB2_GRP1_IsEnabledClock
- AHB2ENR1 SRAM3EN LL_AHB2_GRP1_IsEnabledClock
-

LL_AHB2_GRP1_DisableBusClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableBusClock (void)

Function description

Disable AHB2_1 bus clock.

Return values

- **None:**

Notes

- except for SRAM2 and SRAM3.

Reference Manual to LL API cross reference:

- CFGR2 AHB2DIS1 LL_AHB2_GRP1_DisableBusClock

LL_AHB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB2_GRP1_PERIPH_ALL
  – LL_AHB2_GRP1_PERIPH_GPIOA
  – LL_AHB2_GRP1_PERIPH_GPIOB
  – LL_AHB2_GRP1_PERIPH_GPIOC
  – LL_AHB2_GRP1_PERIPH_GPIOD
  – LL_AHB2_GRP1_PERIPH_GPIOE
  – LL_AHB2_GRP1_PERIPH_GPIOF (*)
  – LL_AHB2_GRP1_PERIPH_GPIOG
  – LL_AHB2_GRP1_PERIPH_GPIOH
  – LL_AHB2_GRP1_PERIPH_GPIOI (*)
  – LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  – LL_AHB2_GRP1_PERIPH_ADC12
  – LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  – LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  – LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  – LL_AHB2_GRP1_PERIPH_USBPHY (*)
  – LL_AHB2_GRP1_PERIPH_AES
  – LL_AHB2_GRP1_PERIPH_HASH
  – LL_AHB2_GRP1_PERIPH_RNG
  – LL_AHB2_GRP1_PERIPH_PKA
  – LL_AHB2_GRP1_PERIPH_SAES (*)
  – LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  – LL_AHB2_GRP1_PERIPH_SDMMC1
  – LL_AHB2_GRP1_PERIPH_SDMMC2 (*)
  – LL_AHB2_GRP1_PERIPH_SRAM2
  – LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR1 GPIOAEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOBEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOCEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIODEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOEEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOFEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOHEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOIEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 GPIOJEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 ADC12EN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 DCMI_PSSIEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 OTGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 USBPHYCEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 AESEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 HASHEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 RNGEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 PKAEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 SAESEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 OSPIMEN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 OTFDEC1EN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 OTFDEC2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 SDMMC1EN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 SDMMC2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 SRAM2EN LL_AHB2_GRP1_DisableClock
- AHB2ENR1 SRAM3EN LL_AHB2_GRP1_DisableClock
-

LL_AHB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB2_GRP1_PERIPH_ALL
  – LL_AHB2_GRP1_PERIPH_GPIOA
  – LL_AHB2_GRP1_PERIPH_GPIOB
  – LL_AHB2_GRP1_PERIPH_GPIOC
  – LL_AHB2_GRP1_PERIPH_GPIOD
  – LL_AHB2_GRP1_PERIPH_GPIOE
  – LL_AHB2_GRP1_PERIPH_GPIOF (*)
  – LL_AHB2_GRP1_PERIPH_GPIOG
  – LL_AHB2_GRP1_PERIPH_GPIOH
  – LL_AHB2_GRP1_PERIPH_GPIOI (*)
  – LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  – LL_AHB2_GRP1_PERIPH_ADC12
  – LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  – LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  – LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  – LL_AHB2_GRP1_PERIPH_USBPHY (*)
  – LL_AHB2_GRP1_PERIPH_AES
  – LL_AHB2_GRP1_PERIPH_HASH
  – LL_AHB2_GRP1_PERIPH_RNG
  – LL_AHB2_GRP1_PERIPH_PKA
  – LL_AHB2_GRP1_PERIPH_SAES (*)
  – LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  – LL_AHB2_GRP1_PERIPH_SDMMC1
  – LL_AHB2_GRP1_PERIPH_SDMMC2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR1 GPIOARST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOBRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOCRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIODRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOERST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOFRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOHRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOIRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 GPIOJRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 ADC12RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 DCMI_PSSIRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 OTGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 USBPHYCRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 AESRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 HASHRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 RNGRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 PKARST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 SAESRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 OCTOSPIMRST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 OTFDEC1RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 OTFDEC2RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 SDMMC1RST LL_AHB2_GRP1_ForceReset
- AHB2RSTR1 SDMMC2RST LL_AHB2_GRP1_ForceReset
-

LL_AHB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP1_PERIPH_ALL
  - LL_AHB2_GRP1_PERIPH_GPIOA
  - LL_AHB2_GRP1_PERIPH_GPIOB
  - LL_AHB2_GRP1_PERIPH_GPIOC
  - LL_AHB2_GRP1_PERIPH_GPIOD
  - LL_AHB2_GRP1_PERIPH_GPIOE
  - LL_AHB2_GRP1_PERIPH_GPIOF (*)
  - LL_AHB2_GRP1_PERIPH_GPIOG
  - LL_AHB2_GRP1_PERIPH_GPIOH
  - LL_AHB2_GRP1_PERIPH_GPIOI (*)
  - LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  - LL_AHB2_GRP1_PERIPH_ADC12
  - LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  - LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  - LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  - LL_AHB2_GRP1_PERIPH_USBPHY (*)
  - LL_AHB2_GRP1_PERIPH_AES
  - LL_AHB2_GRP1_PERIPH_HASH
  - LL_AHB2_GRP1_PERIPH_RNG
  - LL_AHB2_GRP1_PERIPH_PKA
  - LL_AHB2_GRP1_PERIPH_SAES (*)
  - LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  - LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  - LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  - LL_AHB2_GRP1_PERIPH_SDMMC1
  - LL_AHB2_GRP1_PERIPH_SDMMC2 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR1 GPIOARST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOBRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOCRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIODRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOERST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOFRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOHRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOIRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 GPIOJRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 ADC12RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 DCMI_PSSIRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 OTGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 USBPHYCRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 AESRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 HASHRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 RNGRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 PKARST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 SAESRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 OCTOSPIMRST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 OTFDEC1RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 OTFDEC2RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 SDMMC1RST LL_AHB2_GRP1_ReleaseReset
- AHB2RSTR1 SDMMC2RST LL_AHB2_GRP1_ReleaseReset
-

LL_AHB2_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB2_GRP1_PERIPH_ALL
  – LL_AHB2_GRP1_PERIPH_GPIOA
  – LL_AHB2_GRP1_PERIPH_GPIOB
  – LL_AHB2_GRP1_PERIPH_GPIOC
  – LL_AHB2_GRP1_PERIPH_GPIOD
  – LL_AHB2_GRP1_PERIPH_GPIOE
  – LL_AHB2_GRP1_PERIPH_GPIOF (*)
  – LL_AHB2_GRP1_PERIPH_GPIOG
  – LL_AHB2_GRP1_PERIPH_GPIOH
  – LL_AHB2_GRP1_PERIPH_GPIOI (*)
  – LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  – LL_AHB2_GRP1_PERIPH_ADC12
  – LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  – LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  – LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  – LL_AHB2_GRP1_PERIPH_USBPHY (*)
  – LL_AHB2_GRP1_PERIPH_AES
  – LL_AHB2_GRP1_PERIPH_HASH
  – LL_AHB2_GRP1_PERIPH_RNG
  – LL_AHB2_GRP1_PERIPH_PKA
  – LL_AHB2_GRP1_PERIPH_SAES (*)
  – LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  – LL_AHB2_GRP1_PERIPH_SDMMC1
  – LL_AHB2_GRP1_PERIPH_SDMMC2 (*)
  – LL_AHB2_GRP1_PERIPH_SRAM2
  – LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR1 GPIOASMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOBSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOCSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIODSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOESMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOFSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOGSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOHSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOISMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 GPIOJSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 ADC12SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 DCMI_PSSISMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 OTGSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2ENR1 USBPHYCEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 AESSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 HASHSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 RNGSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 PKASMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 SAESSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 OSPIMSMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 OTFDEC1SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 OTFDEC2SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 SDMMC1SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 SDMMC2SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 SRAM2SMEN LL_AHB2_GRP1_EnableClockStopSleep
- AHB2SMENR1 SRAM3SMEN LL_AHB2_GRP1_EnableClockStopSleep
-

LL_AHB2_GRP1_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP1_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if AHB2 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB2_GRP1_PERIPH_ALL
  – LL_AHB2_GRP1_PERIPH_GPIOA
  – LL_AHB2_GRP1_PERIPH_GPIOB
  – LL_AHB2_GRP1_PERIPH_GPIOC
  – LL_AHB2_GRP1_PERIPH_GPIOD
  – LL_AHB2_GRP1_PERIPH_GPIOE
  – LL_AHB2_GRP1_PERIPH_GPIOF (*)
  – LL_AHB2_GRP1_PERIPH_GPIOG
  – LL_AHB2_GRP1_PERIPH_GPIOH
  – LL_AHB2_GRP1_PERIPH_GPIOI (*)
  – LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  – LL_AHB2_GRP1_PERIPH_ADC12
  – LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  – LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  – LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  – LL_AHB2_GRP1_PERIPH_USBPHY (*)
  – LL_AHB2_GRP1_PERIPH_AES
  – LL_AHB2_GRP1_PERIPH_HASH
  – LL_AHB2_GRP1_PERIPH_RNG
  – LL_AHB2_GRP1_PERIPH_PKA
  – LL_AHB2_GRP1_PERIPH_SAES (*)
  – LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  – LL_AHB2_GRP1_PERIPH_SDMMC1
  – LL_AHB2_GRP1_PERIPH_SDMMC2 (*)
  – LL_AHB2_GRP1_PERIPH_SRAM2
  – LL_AHB2_GRP1_PERIPH_SRAM3 (*)
  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2SMENR1 GPIOASMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOBSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOCSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIODSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOESMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOFSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOGSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOHSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOISMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 GPIOJSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 ADC12SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 DCMI_PSSISMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 OTGSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 USBPHYCMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 AESSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 HASHSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 RNGSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 PKASMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 SAESSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 OSPIMSMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 OTFDEC1SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 OTFDEC2SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 SDMMC1SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 SDMMC2SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 SRAM2SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
- AHB2SMENR1 SRAM3SMEN LL_AHB2_GRP1_IsEnabledClockStopSleep
-

LL_AHB2_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB2_GRP1_PERIPH_ALL
  – LL_AHB2_GRP1_PERIPH_GPIOA
  – LL_AHB2_GRP1_PERIPH_GPIOB
  – LL_AHB2_GRP1_PERIPH_GPIOC
  – LL_AHB2_GRP1_PERIPH_GPIOD
  – LL_AHB2_GRP1_PERIPH_GPIOE
  – LL_AHB2_GRP1_PERIPH_GPIOF (*)
  – LL_AHB2_GRP1_PERIPH_GPIOG
  – LL_AHB2_GRP1_PERIPH_GPIOH
  – LL_AHB2_GRP1_PERIPH_GPIOI (*)
  – LL_AHB2_GRP1_PERIPH_GPIOJ (*)
  – LL_AHB2_GRP1_PERIPH_ADC12
  – LL_AHB2_GRP1_PERIPH_DCMI_PSSI
  – LL_AHB2_GRP1_PERIPH_OTG_FS (*)
  – LL_AHB2_GRP1_PERIPH_OTG_HS (*)
  – LL_AHB2_GRP1_PERIPH_USBPHY (*)
  – LL_AHB2_GRP1_PERIPH_AES
  – LL_AHB2_GRP1_PERIPH_HASH
  – LL_AHB2_GRP1_PERIPH_RNG
  – LL_AHB2_GRP1_PERIPH_PKA
  – LL_AHB2_GRP1_PERIPH_SAES (*)
  – LL_AHB2_GRP1_PERIPH_OCTOSPIM (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC1 (*)
  – LL_AHB2_GRP1_PERIPH_OTFDEC2 (*)
  – LL_AHB2_GRP1_PERIPH_SDMMC1
  – LL_AHB2_GRP1_PERIPH_SDMMC2 (*)
  – LL_AHB2_GRP1_PERIPH_SRAM2
  – LL_AHB2_GRP1_PERIPH_SRAM3 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR1 GPIOASMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOBSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOCSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIODSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOESMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOFSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOGSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOHSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOISMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 GPIOJSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 ADC12SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 DCMI_PSSISMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 OTGSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 USBPHYCSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 AESSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 HASHSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 RNGSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 PKASMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 SAESSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 OSPIMSMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 OTFDEC1SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 OTFDEC2SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 SDMMC1SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 SDMMC2SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 SRAM2SMEN LL_AHB2_GRP1_DisableClockStopSleep
- AHB2SMENR1 SRAM3SMEN LL_AHB2_GRP1_DisableClockStopSleep
-

LL_AHB3_GRP1_EnableBusClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableBusClock (void)

Function description

Enable AHB3 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB3DIS LL_AHB3_GRP1_EnableBusClock

LL_AHB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_ALL
  - LL_AHB3_GRP1_PERIPH_LPGPIO1
  - LL_AHB3_GRP1_PERIPH_PWR
  - LL_AHB3_GRP1_PERIPH_ADC4
  - LL_AHB3_GRP1_PERIPH_DAC1
  - LL_AHB3_GRP1_PERIPH_LPDMA1
  - LL_AHB3_GRP1_PERIPH_ADF1
  - LL_AHB3_GRP1_PERIPH_GTZC2
  - LL_AHB3_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR LPGPIO1EN LL_AHB3_GRP1_EnableClock
- AHB3ENR PWREN LL_AHB3_GRP1_EnableClock
- AHB3ENR ADC4EN LL_AHB3_GRP1_EnableClock
- AHB3ENR DAC1EN LL_AHB3_GRP1_EnableClock
- AHB3ENR LPDMA1EN LL_AHB3_GRP1_EnableClock
- AHB3ENR ADF1EN LL_AHB3_GRP1_EnableClock
- AHB3ENR GTZC2EN LL_AHB3_GRP1_EnableClock
- AHB3ENR SRAM4EN LL_AHB3_GRP1_EnableClock
-

LL_AHB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_ALL
  - LL_AHB3_GRP1_PERIPH_LPGPIO1
  - LL_AHB3_GRP1_PERIPH_PWR
  - LL_AHB3_GRP1_PERIPH_ADC4
  - LL_AHB3_GRP1_PERIPH_DAC1
  - LL_AHB3_GRP1_PERIPH_LPDMA1
  - LL_AHB3_GRP1_PERIPH_ADF1
  - LL_AHB3_GRP1_PERIPH_GTZC2
  - LL_AHB3_GRP1_PERIPH_SRAM4

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB3ENR LPGPIO1EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR PWREN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR ADC4EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR DAC1EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR LPDMA1EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR ADF1EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR GTZC2EN LL_AHB3_GRP1_IsEnabledClock
- AHB3ENR SRAM4EN LL_AHB3_GRP1_IsEnabledClock
-

LL_AHB3_GRP1_DisableBusClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableBusClock (void)

Function description

Disable AHB3 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB3DIS LL_AHB3_GRP1_DisableBusClock

LL_AHB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB3_GRP1_PERIPH_ALL
  – LL_AHB3_GRP1_PERIPH_LPGPIO1
  – LL_AHB3_GRP1_PERIPH_PWR
  – LL_AHB3_GRP1_PERIPH_ADC4
  – LL_AHB3_GRP1_PERIPH_DAC1
  – LL_AHB3_GRP1_PERIPH_LPDMA1
  – LL_AHB3_GRP1_PERIPH_ADF1
  – LL_AHB3_GRP1_PERIPH_GTZC2
  – LL_AHB3_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3ENR LPGPIO1EN LL_AHB3_GRP1_DisableClock
- AHB3ENR PWREN LL_AHB3_GRP1_DisableClock
- AHB3ENR ADC4EN LL_AHB3_GRP1_DisableClock
- AHB3ENR DAC1EN LL_AHB3_GRP1_DisableClock
- AHB3ENR LPDMA1EN LL_AHB3_GRP1_DisableClock
- AHB3ENR ADF1EN LL_AHB3_GRP1_DisableClock
- AHB3ENR GTZC2EN LL_AHB3_GRP1_DisableClock
- AHB3ENR SRAM4EN LL_AHB3_GRP1_DisableClock
-

LL_AHB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB3_GRP1_PERIPH_ALL
  - LL_AHB3_GRP1_PERIPH_LPGPIO1
  - LL_AHB3_GRP1_PERIPH_PWR
  - LL_AHB3_GRP1_PERIPH_ADC4
  - LL_AHB3_GRP1_PERIPH_DAC1
  - LL_AHB3_GRP1_PERIPH_LPDMA1
  - LL_AHB3_GRP1_PERIPH_ADF1
  - LL_AHB3_GRP1_PERIPH_GTZC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTR LPGPIO1RST LL_AHB3_GRP1_ForceReset
- AHB3RSTR PWRRST LL_AHB3_GRP1_ForceReset
- AHB3RSTR ADC4RST LL_AHB3_GRP1_ForceReset
- AHB3RSTR DAC1RST LL_AHB3_GRP1_ForceReset
- AHB3RSTR LPDMA1RST LL_AHB3_GRP1_ForceReset
- AHB3RSTR ADF1RST LL_AHB3_GRP1_ForceReset
- AHB3RSTR GTZC2RST LL_AHB3_GRP1_ForceReset
-

LL_AHB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB3_GRP1_PERIPH_ALL
  – LL_AHB3_GRP1_PERIPH_LPGPIO1
  – LL_AHB3_GRP1_PERIPH_PWR
  – LL_AHB3_GRP1_PERIPH_ADC4
  – LL_AHB3_GRP1_PERIPH_DAC1
  – LL_AHB3_GRP1_PERIPH_LPDMA1
  – LL_AHB3_GRP1_PERIPH_ADF1
  – LL_AHB3_GRP1_PERIPH_GTZC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3RSTR LPGPIO1RST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR PWRRST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR ADC4RST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR DAC1RST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR LPDMA1RST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR ADF1RST LL_AHB3_GRP1_ReleaseReset
- AHB3RSTR GTZC2RST LL_AHB3_GRP1_ReleaseReset
-

LL_AHB3_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB3_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB3 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB3_GRP1_PERIPH_ALL
  – LL_AHB3_GRP1_PERIPH_LPGPIO1
  – LL_AHB3_GRP1_PERIPH_PWR
  – LL_AHB3_GRP1_PERIPH_ADC4
  – LL_AHB3_GRP1_PERIPH_DAC1
  – LL_AHB3_GRP1_PERIPH_LPDMA1
  – LL_AHB3_GRP1_PERIPH_ADF1
  – LL_AHB3_GRP1_PERIPH_GTZC2
  – LL_AHB3_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3SMENR LPGPIO1SMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR PWRSMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR ADC4SMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR DAC1SMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR LPDMA1SMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR ADF1SMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR GTZC2SMEN LL_AHB3_GRP1_EnableClockStopSleep
- AHB3SMENR SRAM4SMEN LL_AHB3_GRP1_EnableClockStopSleep
-

LL_AHB3_GRP1_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_AHB3_GRP1_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if AHB3 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB3_GRP1_PERIPH_ALL
  – LL_AHB3_GRP1_PERIPH_LPGPIO1
  – LL_AHB3_GRP1_PERIPH_PWR
  – LL_AHB3_GRP1_PERIPH_ADC4
  – LL_AHB3_GRP1_PERIPH_DAC1
  – LL_AHB3_GRP1_PERIPH_LPDMA1
  – LL_AHB3_GRP1_PERIPH_ADF1
  – LL_AHB3_GRP1_PERIPH_GTZC2
  – LL_AHB3_GRP1_PERIPH_SRAM4

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB3SMENR LPGPIO1SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR PWRSMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR ADC4SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR DAC1SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR LPDMA1SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR ADF1SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR GTZC2SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
- AHB3SMENR SRAM4SMEN LL_AHB3_GRP1_IsEnabledClockStopSleep
-

LL_AHB3_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB3_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB3 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB3_GRP1_PERIPH_ALL
  – LL_AHB3_GRP1_PERIPH_LPGPIO1
  – LL_AHB3_GRP1_PERIPH_PWR
  – LL_AHB3_GRP1_PERIPH_ADC4
  – LL_AHB3_GRP1_PERIPH_DAC1
  – LL_AHB3_GRP1_PERIPH_LPDMA1
  – LL_AHB3_GRP1_PERIPH_ADF1
  – LL_AHB3_GRP1_PERIPH_GTZC2
  – LL_AHB3_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB3SMENR LPGPIO1SMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR PWRSMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR ADC4SMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR DAC1SMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR LPDMA1SMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR ADF1SMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR GTZC2SMEN LL_AHB3_GRP1_DisableClockStopSleep
- AHB3SMENR SRAM4SMEN LL_AHB3_GRP1_DisableClockStopSleep
-

LL_AHB2_GRP2_EnableBusClock

Function name

__STATIC_INLINE void LL_AHB2_GRP2_EnableBusClock (void)

Function description

Enable AHB2_2 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB2DIS2 LL_AHB2_GRP2_EnableBusClock

LL_AHB2_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM6 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM5 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR2 FSMCEN LL_AHB2_GRP2_EnableClock
- AHB2ENR2 OCTOSPI1EN LL_AHB2_GRP2_EnableClock
- AHB2ENR2 OCTOSPI2EN LL_AHB2_GRP2_EnableClock
- AHB2ENR2 HSPI1EN LL_AHB2_GRP2_EnableClock
- AHB2ENR2 SRAM6EN LL_AHB2_GRP2_EnableClock
- AHB2ENR2 SRAM5EN LL_AHB2_GRP2_EnableClock
-

LL_AHB2_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if AHB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM6 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM5 (*)

  (*) value not defined in all devices.

Return values

- **None:**
- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2ENR2 FSMCEN LL_AHB2_GRP2_IsEnabledClock
- AHB2ENR2 OCTOSPI1EN LL_AHB2_GRP2_IsEnabledClock
- AHB2ENR2 OCTOSPI2EN LL_AHB2_GRP2_IsEnabledClock
- AHB2ENR2 HSPI1EN LL_AHB2_GRP2_IsEnabledClock
- AHB2ENR2 SRAM6EN LL_AHB2_GRP2_DisableClock
- AHB2ENR2 SRAM5EN LL_AHB2_GRP2_IsEnabledClock
-

LL_AHB2_GRP2_DisableBusClock

Function name

__STATIC_INLINE void LL_AHB2_GRP2_DisableBusClock (void)

Function description

Disable AHB2_2 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 AHB2DIS2 LL_AHB2_GRP2_DisableBusClock

LL_AHB2_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_AHB2_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable AHB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB2_GRP2_PERIPH_ALL
  – LL_AHB2_GRP2_PERIPH_FSMC (*)
  – LL_AHB2_GRP2_PERIPH_OCTOSPI1
  – LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  – LL_AHB2_GRP2_PERIPH_HSPI1 (*)
  – LL_AHB2_GRP2_PERIPH_SRAM6 (*)
  – LL_AHB2_GRP2_PERIPH_SRAM5 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2ENR2 FSMCEN LL_AHB2_GRP2_DisableClock
- AHB2ENR2 OCTOSPI1EN LL_AHB2_GRP2_DisableClock
- AHB2ENR2 OCTOSPI2EN LL_AHB2_GRP2_DisableClock
- AHB2ENR2 HSPI1EN LL_AHB2_GRP2_DisableClock
- AHB2ENR2 SRAM6EN LL_AHB2_GRP2_DisableClock
- AHB2ENR2 SRAM5EN LL_AHB2_GRP2_DisableClock
-

LL_AHB2_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_AHB2_GRP2_ForceReset (uint32_t Periphs)

Function description

Force AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR2 FSMCRST LL_AHB2_GRP2_ForceReset
- AHB2RSTR2 OCTOSPI1RST LL_AHB2_GRP2_ForceReset
- AHB2RSTR2 OCTOSPI2RST LL_AHB2_GRP2_ForceReset
- AHB2RSTR2 HSPI1RST LL_AHB2_GRP2_ForceReset
-

LL_AHB2_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB2_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release AHB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2RSTR2 FSMCRST LL_AHB2_GRP2_ReleaseReset
- AHB2RSTR2 OCTOSPI1RST LL_AHB2_GRP2_ReleaseReset
- AHB2RSTR2 OCTOSPI2RST LL_AHB2_GRP2_ReleaseReset
- AHB2RSTR2 HSPI1RST LL_AHB2_GRP2_ReleaseReset
-

LL_AHB2_GRP2_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP2_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM6 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM5 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR2 FSMCSMEN LL_AHB2_GRP2_EnableClockStopSleep
- AHB2SMENR2 OCTOSPI1SMEN LL_AHB2_GRP2_EnableClockStopSleep
- AHB2SMENR2 OCTOSPI2SMEN LL_AHB2_GRP2_EnableClockStopSleep
- AHB2SMENR2 HSPI1SMEN LL_AHB2_GRP2_EnableClockStopSleep
- AHB2SMENR2 SRAM6SMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
- AHB2SMENR2 SRAM5SMEN LL_AHB2_GRP2_EnableClockStopSleep
-

LL_AHB2_GRP2_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_AHB2_GRP2_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if AHB2 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM6 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM5 (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHB2SMENR2 FSMCSMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
- AHB2SMENR2 OCTOSPI1SMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
- AHB2SMENR2 OCTOSPI2SMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
- AHB2SMENR2 HSPI1SMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
- AHB2SMENR2 SRAM6SMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
- AHB2SMENR2 SRAM5SMEN LL_AHB2_GRP2_IsEnabledClockStopSleep
-

LL_AHB2_GRP2_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_AHB2_GRP2_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable AHB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB2_GRP2_PERIPH_ALL
  - LL_AHB2_GRP2_PERIPH_FSMC (*)
  - LL_AHB2_GRP2_PERIPH_OCTOSPI1
  - LL_AHB2_GRP2_PERIPH_OCTOSPI2 (*)
  - LL_AHB2_GRP2_PERIPH_HSPI1 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM6 (*)
  - LL_AHB2_GRP2_PERIPH_SRAM5 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHB2SMENR2 FSMCSMEN LL_AHB2_GRP2_DisableClockStopSleep
- AHB2SMENR2 OCTOSPI1SMEN LL_AHB2_GRP2_DisableClockStopSleep
- AHB2SMENR2 OCTOSPI2SMEN LL_AHB2_GRP2_DisableClockStopSleep
- AHB2SMENR2 HSPI1SMEN LL_AHB2_GRP2_DisableClockStopSleep
- AHB2SMENR2 SRAM6SMEN LL_AHB2_GRP2_DisableClockStopSleep
- AHB2SMENR2 SRAM5SMEN LL_AHB2_GRP2_DisableClockStopSleep
-

LL_APB1_GRP1_EnableBusClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableBusClock (void)

Function description

Enable APB1 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB1DIS LL_APB1_GRP1_EnableBusClock

LL_APB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_USART2 (*)
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_EnableClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_EnableClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_EnableClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART3EN LL_APB1_GRP1_EnableClock
- APB1ENR1 UART4EN LL_APB1_GRP1_EnableClock
- APB1ENR1 UART5EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_EnableClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_EnableClock
- APB1ENR1 CRSEN LL_APB1_GRP1_EnableClock
- APB1ENR1 USART6N LL_APB1_GRP1_EnableClock
-

LL_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR2 I2C4EN LL_APB1_GRP2_EnableClock
- APB1ENR2 I2C5EN LL_APB1_GRP2_EnableClock
- APB1ENR2 I2C6EN LL_APB1_GRP2_EnableClock
- APB1ENR2 LPTIM2EN LL_APB1_GRP2_EnableClock
- APB1ENR2 FDCAN1EN LL_APB1_GRP2_EnableClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_EnableClock

LL_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_USART2 (*)
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART3EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART4EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 UART5EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 CRSEN LL_APB1_GRP1_IsEnabledClock
- APB1ENR1 USART6EN LL_APB1_GRP1_IsEnabledClock
-

LL_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1ENR2 I2C4EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 I2C5EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 I2C6EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 LPTIM2EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 FDCAN1EN LL_APB1_GRP2_IsEnabledClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_IsEnabledClock

LL_APB1_GRP1_DisableBusClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableBusClock (void)

Function description

Disable APB1 bus clock.

Return values

- **None:**

Notes

- except for IWDG.

Reference Manual to LL API cross reference:

- CFGR2 APB1DIS LL_APB1_GRP1_DisableBusClock

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP1_PERIPH_ALL
  – LL_APB1_GRP1_PERIPH_TIM2
  – LL_APB1_GRP1_PERIPH_TIM3
  – LL_APB1_GRP1_PERIPH_TIM4
  – LL_APB1_GRP1_PERIPH_TIM5
  – LL_APB1_GRP1_PERIPH_TIM6
  – LL_APB1_GRP1_PERIPH_TIM7
  – LL_APB1_GRP1_PERIPH_WWDG
  – LL_APB1_GRP1_PERIPH_SPI2
  – LL_APB1_GRP1_PERIPH_USART2 (*)
  – LL_APB1_GRP1_PERIPH_USART3
  – LL_APB1_GRP1_PERIPH_UART4
  – LL_APB1_GRP1_PERIPH_UART5
  – LL_APB1_GRP1_PERIPH_I2C1
  – LL_APB1_GRP1_PERIPH_I2C2
  – LL_APB1_GRP1_PERIPH_CRS
  – LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR1 TIM2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM3EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM4EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM5EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM6EN LL_APB1_GRP1_DisableClock
- APB1ENR1 TIM7EN LL_APB1_GRP1_DisableClock
- APB1ENR1 WWDGEN LL_APB1_GRP1_DisableClock
- APB1ENR1 SPI2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 USART2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 USART3EN LL_APB1_GRP1_DisableClock
- APB1ENR1 UART4EN LL_APB1_GRP1_DisableClock
- APB1ENR1 UART5EN LL_APB1_GRP1_DisableClock
- APB1ENR1 I2C1EN LL_APB1_GRP1_DisableClock
- APB1ENR1 I2C2EN LL_APB1_GRP1_DisableClock
- APB1ENR1 CRSEN LL_APB1_GRP1_DisableClock
- APB1ENR1 USART6EN LL_APB1_GRP1_DisableClock
-

LL_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1ENR2 I2C4EN LL_APB1_GRP2_DisableClock
- APB1ENR2 I2C5EN LL_APB1_GRP2_DisableClock
- APB1ENR2 I2C6EN LL_APB1_GRP2_DisableClock
- APB1ENR2 LPTIM2EN LL_APB1_GRP2_DisableClock
- APB1ENR2 FDCAN1EN LL_APB1_GRP2_DisableClock
- APB1ENR2 UCPD1EN LL_APB1_GRP2_DisableClock
-

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP1_PERIPH_ALL
  – LL_APB1_GRP1_PERIPH_TIM2
  – LL_APB1_GRP1_PERIPH_TIM3
  – LL_APB1_GRP1_PERIPH_TIM4
  – LL_APB1_GRP1_PERIPH_TIM5
  – LL_APB1_GRP1_PERIPH_TIM6
  – LL_APB1_GRP1_PERIPH_TIM7
  – LL_APB1_GRP1_PERIPH_SPI2
  – LL_APB1_GRP1_PERIPH_USART2 (*)
  – LL_APB1_GRP1_PERIPH_USART3
  – LL_APB1_GRP1_PERIPH_UART4
  – LL_APB1_GRP1_PERIPH_UART5
  – LL_APB1_GRP1_PERIPH_I2C1
  – LL_APB1_GRP1_PERIPH_I2C2
  – LL_APB1_GRP1_PERIPH_CRS
  – LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR1 TIM2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM4RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM5RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM6RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 TIM7RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 SPI2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART3RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 UART4RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 UART5RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C1RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 I2C2RST LL_APB1_GRP1_ForceReset
- APB1RSTR1 CRSRST LL_APB1_GRP1_ForceReset
- APB1RSTR1 USART6RST LL_APB1_GRP1_ForceReset
-

LL_APB1_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR2 I2C4RST LL_APB1_GRP2_DisableClock
- APB1RSTR2 I2C5RST LL_APB1_GRP2_DisableClock
- APB1RSTR2 I2C6RST LL_APB1_GRP2_DisableClock
- APB1RSTR2 LPTIM2RST LL_APB1_GRP2_DisableClock
- APB1RSTR2 FDCAN1RST LL_APB1_GRP2_DisableClock
- APB1RSTR2 UCPDRST LL_APB1_GRP2_DisableClock
-

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP1_PERIPH_ALL
  – LL_APB1_GRP1_PERIPH_TIM2
  – LL_APB1_GRP1_PERIPH_TIM3
  – LL_APB1_GRP1_PERIPH_TIM4
  – LL_APB1_GRP1_PERIPH_TIM5
  – LL_APB1_GRP1_PERIPH_TIM6
  – LL_APB1_GRP1_PERIPH_TIM7
  – LL_APB1_GRP1_PERIPH_SPI2
  – LL_APB1_GRP1_PERIPH_USART2 (*)
  – LL_APB1_GRP1_PERIPH_USART3
  – LL_APB1_GRP1_PERIPH_UART4
  – LL_APB1_GRP1_PERIPH_UART5
  – LL_APB1_GRP1_PERIPH_I2C1
  – LL_APB1_GRP1_PERIPH_I2C2
  – LL_APB1_GRP1_PERIPH_CRS
  – LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR1 TIM2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM6RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 TIM7RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 SPI2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 USART2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 USART3RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 UART4RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 UART5RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 I2C1RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 I2C2RST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 CRSRST LL_APB1_GRP1_ReleaseReset
- APB1RSTR1 USART6RST LL_APB1_GRP1_ReleaseReset
-

LL_APB1_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)
  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1RSTR2 I2C4RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 I2C5RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 I2C6RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 LPTIM2RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 FDCAN1RST LL_APB1_GRP2_ReleaseReset
- APB1RSTR2 UCPD1RST LL_APB1_GRP2_ReleaseReset
-

LL_APB1_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP1_PERIPH_ALL
  – LL_APB1_GRP1_PERIPH_TIM2
  – LL_APB1_GRP1_PERIPH_TIM3
  – LL_APB1_GRP1_PERIPH_TIM4
  – LL_APB1_GRP1_PERIPH_TIM5
  – LL_APB1_GRP1_PERIPH_TIM6
  – LL_APB1_GRP1_PERIPH_TIM7
  – LL_APB1_GRP1_PERIPH_SPI2
  – LL_APB1_GRP1_PERIPH_USART2 (*)
  – LL_APB1_GRP1_PERIPH_USART3
  – LL_APB1_GRP1_PERIPH_UART4
  – LL_APB1_GRP1_PERIPH_UART5
  – LL_APB1_GRP1_PERIPH_I2C1
  – LL_APB1_GRP1_PERIPH_I2C2
  – LL_APB1_GRP1_PERIPH_CRS
  – LL_APB1_GRP1_PERIPH_USART6 (*)
  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_EnableClockStopSleep
- APB1SMENR1 USART6SMEN LL_APB1_GRP1_EnableClockStopSleep
-

LL_APB1_GRP1_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if APB1 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP1_PERIPH_ALL
  – LL_APB1_GRP1_PERIPH_TIM2
  – LL_APB1_GRP1_PERIPH_TIM3
  – LL_APB1_GRP1_PERIPH_TIM4
  – LL_APB1_GRP1_PERIPH_TIM5
  – LL_APB1_GRP1_PERIPH_TIM6
  – LL_APB1_GRP1_PERIPH_TIM7
  – LL_APB1_GRP1_PERIPH_SPI2
  – LL_APB1_GRP1_PERIPH_USART2 (*)
  – LL_APB1_GRP1_PERIPH_USART3
  – LL_APB1_GRP1_PERIPH_UART4
  – LL_APB1_GRP1_PERIPH_UART5
  – LL_APB1_GRP1_PERIPH_I2C1
  – LL_APB1_GRP1_PERIPH_I2C2
  – LL_APB1_GRP1_PERIPH_CRS
  – LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_IsEnabledClockStopSleep
- APB1SMENR1 USART6SMEN LL_APB1_GRP1_IsEnabledClockStopSleep
-

LL_APB1_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM2
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_TIM4
  - LL_APB1_GRP1_PERIPH_TIM5
  - LL_APB1_GRP1_PERIPH_TIM6
  - LL_APB1_GRP1_PERIPH_TIM7
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_USART2 (*)
  - LL_APB1_GRP1_PERIPH_USART3
  - LL_APB1_GRP1_PERIPH_UART4
  - LL_APB1_GRP1_PERIPH_UART5
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2
  - LL_APB1_GRP1_PERIPH_CRS
  - LL_APB1_GRP1_PERIPH_USART6 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR1 TIM2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM4SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM5SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM6SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 TIM7SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 SPI2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 UART4SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 UART5SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C1SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 I2C2SMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 CRSSMEN LL_APB1_GRP1_DisableClockStopSleep
- APB1SMENR1 USART6SMEN LL_APB1_GRP1_DisableClockStopSleep
-

LL_APB1_GRP2_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR2 I2C4SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 I2C5SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 I2C6SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 LPTIM2SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 FDCAN1SMEN LL_APB1_GRP2_EnableClockStopSleep
- APB1SMENR2 UCPD1SMEN LL_APB1_GRP2_EnableClockStopSleep
-

LL_APB1_GRP2_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if APB1 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB1SMENR2 I2C4SMEN LL_APB1_GRP2_IsEnabledClockStopSleep
- APB1SMENR2 I2C5SMEN LL_APB1_GRP2_IsEnabledClockStopSleep
- APB1SMENR2 I2C6SMEN LL_APB1_GRP2_IsEnabledClockStopSleep
- APB1SMENR2 LPTIM2SMEN LL_APB1_GRP2_IsEnabledClockStopSleep
- APB1SMENR2 FDCAN1SMEN LL_APB1_GRP2_IsEnabledClockStopSleep
- APB1SMENR2 UCPD1SMEN LL_APB1_GRP2_IsEnabledClockStopSleep

LL_APB1_GRP2_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP2_PERIPH_ALL
  – LL_APB1_GRP2_PERIPH_I2C4
  – LL_APB1_GRP2_PERIPH_I2C5 (*)
  – LL_APB1_GRP2_PERIPH_I2C6 (*)
  – LL_APB1_GRP2_PERIPH_LPTIM2
  – LL_APB1_GRP2_PERIPH_FDCAN1
  – LL_APB1_GRP2_PERIPH_UCPD1 (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB1SMENR2 I2C4SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 I2C5SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 I2C6SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 LPTIM2SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 FDCAN1SMEN LL_APB1_GRP2_DisableClockStopSleep
- APB1SMENR2 UCPD1SMEN LL_APB1_GRP2_DisableClockStopSleep

LL_APB2_GRP1_EnableBusClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableBusClock (void)

Function description

Enable APB2 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB2DIS LL_APB2_GRP1_EnableBusClock

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_ALL
  – LL_APB2_GRP1_PERIPH_TIM1
  – LL_APB2_GRP1_PERIPH_SPI1
  – LL_APB2_GRP1_PERIPH_TIM8
  – LL_APB2_GRP1_PERIPH_USART1
  – LL_APB2_GRP1_PERIPH_TIM15
  – LL_APB2_GRP1_PERIPH_TIM16
  – LL_APB2_GRP1_PERIPH_TIM17
  – LL_APB2_GRP1_PERIPH_SAI1
  – LL_APB2_GRP1_PERIPH_SAI2 (*)
  – LL_APB2_GRP1_PERIPH_USB_FS (*)
  – LL_APB2_GRP1_PERIPH_GFXTIM (*)
  – LL_APB2_GRP1_PERIPH_LTDC (*)
  – LL_APB2_GRP1_PERIPH_DSI (*)
  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_EnableClock
- APB2ENR SPI1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM8EN LL_APB2_GRP1_EnableClock
- APB2ENR USART1EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM15EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM16EN LL_APB2_GRP1_EnableClock
- APB2ENR TIM17EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI1EN LL_APB2_GRP1_EnableClock
- APB2ENR SAI2EN LL_APB2_GRP1_EnableClock
- APB2ENR USBEN LL_APB2_GRP1_EnableClock
- APB2ENR GFXTIMEN LL_APB2_GRP1_EnableClock
- APB2ENR LTDCEN LL_APB2_GRP1_EnableClock
- APB2ENR DSIHOSTEN LL_APB2_GRP1_EnableClock
-

LL_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_ALL
  – LL_APB2_GRP1_PERIPH_TIM1
  – LL_APB2_GRP1_PERIPH_SPI1
  – LL_APB2_GRP1_PERIPH_TIM8
  – LL_APB2_GRP1_PERIPH_USART1
  – LL_APB2_GRP1_PERIPH_TIM15
  – LL_APB2_GRP1_PERIPH_TIM16
  – LL_APB2_GRP1_PERIPH_TIM17
  – LL_APB2_GRP1_PERIPH_SAI1
  – LL_APB2_GRP1_PERIPH_SAI2 (*)
  – LL_APB2_GRP1_PERIPH_USB_FS (*)
  – LL_APB2_GRP1_PERIPH_GFXTIM (*)
  – LL_APB2_GRP1_PERIPH_LTDC (*)
  – LL_APB2_GRP1_PERIPH_DSI (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SPI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM8EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USART1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM15EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM16EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR TIM17EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI1EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR SAI2EN LL_APB2_GRP1_IsEnabledClock
- APB2ENR USBEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR GFXTIMEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR LTDCEN LL_APB2_GRP1_IsEnabledClock
- APB2ENR DSIHOSTEN LL_APB2_GRP1_IsEnabledClock
-

LL_APB2_GRP1_DisableBusClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableBusClock (void)

Function description

Disable APB2 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB2DIS LL_APB2_GRP1_DisableBusClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_ALL
  – LL_APB2_GRP1_PERIPH_TIM1
  – LL_APB2_GRP1_PERIPH_SPI1
  – LL_APB2_GRP1_PERIPH_TIM8
  – LL_APB2_GRP1_PERIPH_USART1
  – LL_APB2_GRP1_PERIPH_TIM15
  – LL_APB2_GRP1_PERIPH_TIM16
  – LL_APB2_GRP1_PERIPH_TIM17
  – LL_APB2_GRP1_PERIPH_SAI1
  – LL_APB2_GRP1_PERIPH_SAI2 (*)
  – LL_APB2_GRP1_PERIPH_USB_FS (*)
  – LL_APB2_GRP1_PERIPH_GFXTIM (*)
  – LL_APB2_GRP1_PERIPH_LTDC (*)
  – LL_APB2_GRP1_PERIPH_DSI (*)
  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR TIM1EN LL_APB2_GRP1_DisableClock
- APB2ENR SPI1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM8EN LL_APB2_GRP1_DisableClock
- APB2ENR USART1EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM15EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM16EN LL_APB2_GRP1_DisableClock
- APB2ENR TIM17EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI1EN LL_APB2_GRP1_DisableClock
- APB2ENR SAI2EN LL_APB2_GRP1_DisableClock
- APB2ENR USBEN LL_APB2_GRP1_DisableClock
- APB2ENR GFXTIMEN LL_APB2_GRP1_DisableClock
- APB2ENR LTDCEN LL_APB2_GRP1_DisableClock
- APB2ENR DSIHOSTEN LL_APB2_GRP1_DisableClock
-

LL_APB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB_FS (*)
  - LL_APB2_GRP1_PERIPH_GFXTIM (*)
  - LL_APB2_GRP1_PERIPH_LTDC (*)
  - LL_APB2_GRP1_PERIPH_DSI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR TIM1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ForceReset
- APB2RSTR USART1RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ForceReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ForceReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ForceReset
- APB2RSTR USBRST LL_APB2_GRP1_ForceReset
- APB2RSTR GFXTIMRST LL_APB2_GRP1_ForceReset
- APB2RSTR LTDCRST LL_APB2_GRP1_ForceReset
- APB2RSTR DSIHOSTRST LL_APB2_GRP1_ForceReset
-

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_ALL
  – LL_APB2_GRP1_PERIPH_TIM1
  – LL_APB2_GRP1_PERIPH_SPI1
  – LL_APB2_GRP1_PERIPH_TIM8
  – LL_APB2_GRP1_PERIPH_USART1
  – LL_APB2_GRP1_PERIPH_TIM15
  – LL_APB2_GRP1_PERIPH_TIM16
  – LL_APB2_GRP1_PERIPH_TIM17
  – LL_APB2_GRP1_PERIPH_SAI1
  – LL_APB2_GRP1_PERIPH_SAI2 (*)
  – LL_APB2_GRP1_PERIPH_USB_FS (*)
  – LL_APB2_GRP1_PERIPH_GFXTIM (*)
  – LL_APB2_GRP1_PERIPH_LTDC (*)
  – LL_APB2_GRP1_PERIPH_DSI (*)
  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2RSTR TIM1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SPI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM8RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USART1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM15RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM16RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR TIM17RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI1RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR SAI2RST LL_APB2_GRP1_ReleaseReset
- APB2RSTR USBRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR GFXTIMRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR LTDCRST LL_APB2_GRP1_ReleaseReset
- APB2RSTR DSIHOSTRST LL_APB2_GRP1_ReleaseReset
-

LL_APB2_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_ALL
  – LL_APB2_GRP1_PERIPH_TIM1
  – LL_APB2_GRP1_PERIPH_SPI1
  – LL_APB2_GRP1_PERIPH_TIM8
  – LL_APB2_GRP1_PERIPH_USART1
  – LL_APB2_GRP1_PERIPH_TIM15
  – LL_APB2_GRP1_PERIPH_TIM16
  – LL_APB2_GRP1_PERIPH_TIM17
  – LL_APB2_GRP1_PERIPH_SAI1
  – LL_APB2_GRP1_PERIPH_SAI2 (*)
  – LL_APB2_GRP1_PERIPH_USB_FS (*)
  – LL_APB2_GRP1_PERIPH_GFXTIM(*)
  – LL_APB2_GRP1_PERIPH_LTDC (*)
  – LL_APB2_GRP1_PERIPH_DSI (*)

  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2SMENR TIM1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SPI1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM8SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR USART1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM15SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM16SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR TIM17SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SAI2SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR USBSMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR SAI2SMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR GFXTIMSMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR LTDCSMEN LL_APB2_GRP1_EnableClockStopSleep
- APB2SMENR DSIHOSTSMEN LL_APB2_GRP1_EnableClockStopSleep
-

LL_APB2_GRP1_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if APB2 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB2_GRP1_PERIPH_ALL
  - LL_APB2_GRP1_PERIPH_TIM1
  - LL_APB2_GRP1_PERIPH_SPI1
  - LL_APB2_GRP1_PERIPH_TIM8
  - LL_APB2_GRP1_PERIPH_USART1
  - LL_APB2_GRP1_PERIPH_TIM15
  - LL_APB2_GRP1_PERIPH_TIM16
  - LL_APB2_GRP1_PERIPH_TIM17
  - LL_APB2_GRP1_PERIPH_SAI1
  - LL_APB2_GRP1_PERIPH_SAI2 (*)
  - LL_APB2_GRP1_PERIPH_USB_FS (*)
  - LL_APB2_GRP1_PERIPH_GFXTIM (*)
  - LL_APB2_GRP1_PERIPH_LTDC (*)
  - LL_APB2_GRP1_PERIPH_DSI (*)

  (*) value not defined in all devices.

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB2SMENR TIM1SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR SPI1SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR TIM8SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR USART1SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR TIM15SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR TIM16SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR TIM17SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR SAI2SMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR USBSMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR GFXTIMSMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR LTDCSMEN LL_APB2_GRP1_IsEnabledClockStopSleep
- APB2SMENR DSIHOSTSMEN LL_APB2_GRP1_IsEnabledClockStopSleep
-

LL_APB2_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_ALL
  – LL_APB2_GRP1_PERIPH_TIM1
  – LL_APB2_GRP1_PERIPH_SPI1
  – LL_APB2_GRP1_PERIPH_TIM8
  – LL_APB2_GRP1_PERIPH_USART1
  – LL_APB2_GRP1_PERIPH_TIM15
  – LL_APB2_GRP1_PERIPH_TIM16
  – LL_APB2_GRP1_PERIPH_TIM17
  – LL_APB2_GRP1_PERIPH_SAI1
  – LL_APB2_GRP1_PERIPH_SAI2 (*)
  – LL_APB2_GRP1_PERIPH_USB_FS (*)
  – LL_APB2_GRP1_PERIPH_GFXTIM (*)
  – LL_APB2_GRP1_PERIPH_LTDC (*)
  – LL_APB2_GRP1_PERIPH_DSI (*)
  (*) value not defined in all devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2SMENR TIM1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SPI1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM8SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR USART1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM15SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM16SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR TIM17SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SAI1SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR SAI2SMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR USBSMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR GFXTIMSMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR LTDCSMEN LL_APB2_GRP1_DisableClockStopSleep
- APB2SMENR DSIHOSTSMEN LL_APB2_GRP1_DisableClockStopSleep
-

LL_APB3_GRP1_EnableBusClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableBusClock (void)

Function description

Enable APB3 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB3DIS LL_APB3_GRP1_EnableBusClock

LL_APB3_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB3 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SYSCFG
  - LL_APB3_GRP1_PERIPH_SPI3
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM4
  - LL_APB3_GRP1_PERIPH_LPTIM4
  - LL_APB3_GRP1_PERIPH_OPAMP
  - LL_APB3_GRP1_PERIPH_COMP
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR SYSCFGEN LL_APB3_GRP1_EnableClock
- APB3ENR SPI3EN LL_APB3_GRP1_EnableClock
- APB3ENR LPUART1EN LL_APB3_GRP1_EnableClock
- APB3ENR I2C3EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM1EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM3EN LL_APB3_GRP1_EnableClock
- APB3ENR LPTIM4EN LL_APB3_GRP1_EnableClock
- APB3ENR OPAMPEN LL_APB3_GRP1_EnableClock
- APB3ENR COMPEN LL_APB3_GRP1_EnableClock
- APB3ENR VREFEN LL_APB3_GRP1_EnableClock
- APB3ENR RTCAPBEN LL_APB3_GRP1_EnableClock
-

LL_APB3_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB3_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB3 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB3_GRP1_PERIPH_ALL
  – LL_APB3_GRP1_PERIPH_SYSCFG
  – LL_APB3_GRP1_PERIPH_SPI3
  – LL_APB3_GRP1_PERIPH_LPUART1
  – LL_APB3_GRP1_PERIPH_I2C3
  – LL_APB3_GRP1_PERIPH_LPTIM1
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_OPAMP
  – LL_APB3_GRP1_PERIPH_COMP
  – LL_APB3_GRP1_PERIPH_VREF
  – LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB3ENR SYSCFGEN LL_APB3_GRP1_IsEnabledClock
- APB3ENR SPI3EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPUART1EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR I2C3EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM1EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM3EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR LPTIM4EN LL_APB3_GRP1_IsEnabledClock
- APB3ENR OPAMPEN LL_APB3_GRP1_IsEnabledClock
- APB3ENR COMPEN LL_APB3_GRP1_IsEnabledClock
- APB3ENR VREFEN LL_APB3_GRP1_IsEnabledClock
- APB3ENR RTCAPBEN LL_APB3_GRP1_IsEnabledClock
-

LL_APB3_GRP1_DisableBusClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableBusClock (void)

Function description

Disable APB3 bus clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 APB3DIS LL_APB3_GRP1_DisableBusClock

LL_APB3_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB3_GRP1_PERIPH_ALL
  – LL_APB3_GRP1_PERIPH_SYSCFG
  – LL_APB3_GRP1_PERIPH_SPI3
  – LL_APB3_GRP1_PERIPH_LPUART1
  – LL_APB3_GRP1_PERIPH_I2C3
  – LL_APB3_GRP1_PERIPH_LPTIM1
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_OPAMP
  – LL_APB3_GRP1_PERIPH_COMP
  – LL_APB3_GRP1_PERIPH_VREF
  – LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3ENR SYSCFGEN LL_APB3_GRP1_DisableClock
- APB3ENR SPI3EN LL_APB3_GRP1_DisableClock
- APB3ENR LPUART1EN LL_APB3_GRP1_DisableClock
- APB3ENR I2C3EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM1EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM3EN LL_APB3_GRP1_DisableClock
- APB3ENR LPTIM4EN LL_APB3_GRP1_DisableClock
- APB3ENR OPAMPEN LL_APB3_GRP1_DisableClock
- APB3ENR COMPEN LL_APB3_GRP1_DisableClock
- APB3ENR VREFEN LL_APB3_GRP1_DisableClock
- APB3ENR RTCAPBEN LL_APB3_GRP1_DisableClock
-

LL_APB3_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB3_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB3_GRP1_PERIPH_ALL
  – LL_APB3_GRP1_PERIPH_SYSCFG
  – LL_APB3_GRP1_PERIPH_SPI3
  – LL_APB3_GRP1_PERIPH_LPUART1
  – LL_APB3_GRP1_PERIPH_I2C3
  – LL_APB3_GRP1_PERIPH_LPTIM1
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_OPAMP
  – LL_APB3_GRP1_PERIPH_COMP
  – LL_APB3_GRP1_PERIPH_VREF
  – LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3RSTR SYSCFGRST LL_APB3_GRP1_ForceReset
- APB3RSTR SPI3RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPUART1RST LL_APB3_GRP1_ForceReset
- APB3RSTR I2C3RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM1RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM3RST LL_APB3_GRP1_ForceReset
- APB3RSTR LPTIM4RST LL_APB3_GRP1_ForceReset
- APB3RSTR OPAMPRST LL_APB3_GRP1_ForceReset
- APB3RSTR COMPRST LL_APB3_GRP1_ForceReset
- APB3RSTR VREFRST LL_APB3_GRP1_ForceReset
- APB3RSTR RTCAPBRST LL_APB3_GRP1_ForceReset
-

LL_APB3_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB3_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB3 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB3_GRP1_PERIPH_ALL
  – LL_APB3_GRP1_PERIPH_SYSCFG
  – LL_APB3_GRP1_PERIPH_SPI3
  – LL_APB3_GRP1_PERIPH_LPUART1
  – LL_APB3_GRP1_PERIPH_I2C3
  – LL_APB3_GRP1_PERIPH_LPTIM1
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_OPAMP
  – LL_APB3_GRP1_PERIPH_COMP
  – LL_APB3_GRP1_PERIPH_VREF
  – LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3RSTR SYSCFGRST LL_APB3_GRP1_ReleaseReset
- APB3RSTR SPI3RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPUART1RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR I2C3RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM1RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM3RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR LPTIM4RST LL_APB3_GRP1_ReleaseReset
- APB3RSTR OPAMPRST LL_APB3_GRP1_ReleaseReset
- APB3RSTR COMPRST LL_APB3_GRP1_ReleaseReset
- APB3RSTR VREFRST LL_APB3_GRP1_ReleaseReset
- APB3RSTR RTCAPBRST LL_APB3_GRP1_ReleaseReset
-

LL_APB3_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB3_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB3 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB3_GRP1_PERIPH_ALL
  – LL_APB3_GRP1_PERIPH_SYSCFG
  – LL_APB3_GRP1_PERIPH_SPI3
  – LL_APB3_GRP1_PERIPH_LPUART1
  – LL_APB3_GRP1_PERIPH_I2C3
  – LL_APB3_GRP1_PERIPH_LPTIM1
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_LPTIM4
  – LL_APB3_GRP1_PERIPH_OPAMP
  – LL_APB3_GRP1_PERIPH_COMP
  – LL_APB3_GRP1_PERIPH_VREF
  – LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3SMENR SYSCFGSMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR SPI3SMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR LPUART1SMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR I2C3SMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR LPTIM1SMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR LPTIM3SMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR LPTIM4SMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR OPAMPSMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR COMPSMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR VREFSMEN LL_APB3_GRP1_EnableClockStopSleep
- APB3SMENR RTCAPBSMEN LL_APB3_GRP1_EnableClockStopSleep
-

LL_APB3_GRP1_IsEnabledClockStopSleep

Function name

__STATIC_INLINE uint32_t LL_APB3_GRP1_IsEnabledClockStopSleep (uint32_t Periphs)

Function description

Check if APB3 peripheral clocks in Sleep and Stop modes is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SYSCFG
  - LL_APB3_GRP1_PERIPH_SPI3
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM4
  - LL_APB3_GRP1_PERIPH_LPTIM4
  - LL_APB3_GRP1_PERIPH_OPAMP
  - LL_APB3_GRP1_PERIPH_COMP
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- APB3SMENR SYSCFGSMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR SPI3SMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR LPUART1SMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR I2C3SMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR LPTIM1SMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR LPTIM3SMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR LPTIM4SMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR OPAMPSMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR COMPSMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR VREFSMEN LL_APB3_GRP1_IsEnabledClockStopSleep
- APB3SMENR RTCAPBSMEN LL_APB3_GRP1_IsEnabledClockStopSleep
-

LL_APB3_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB3_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB3 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB3_GRP1_PERIPH_ALL
  - LL_APB3_GRP1_PERIPH_SYSCFG
  - LL_APB3_GRP1_PERIPH_SPI3
  - LL_APB3_GRP1_PERIPH_LPUART1
  - LL_APB3_GRP1_PERIPH_I2C3
  - LL_APB3_GRP1_PERIPH_LPTIM1
  - LL_APB3_GRP1_PERIPH_LPTIM4
  - LL_APB3_GRP1_PERIPH_LPTIM4
  - LL_APB3_GRP1_PERIPH_OPAMP
  - LL_APB3_GRP1_PERIPH_COMP
  - LL_APB3_GRP1_PERIPH_VREF
  - LL_APB3_GRP1_PERIPH_RTCAPB

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB3SMENR SYSCFGSMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR SPI3SMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR LPUART1SMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR I2C3SMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR LPTIM1SMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR LPTIM3SMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR LPTIM4SMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR OPAMPSMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR COMPSMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR VREFSMEN LL_APB3_GRP1_DisableClockStopSleep
- APB3SMENR RTCAPBSMEN LL_APB3_GRP1_DisableClockStopSleep
-

LL_SRDAMR_GRP1_EnableAutonomousClock

Function name

__STATIC_INLINE void LL_SRDAMR_GRP1_EnableAutonomousClock (uint32_t Periphs)

Function description

Enable SRDAMR peripheral clocks in autonomous mode.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_SRDAMR_GRP1_PERIPH_ALL
  – LL_SRDAMR_GRP1_PERIPH_SPI3
  – LL_SRDAMR_GRP1_PERIPH_LPUART1
  – LL_SRDAMR_GRP1_PERIPH_I2C3
  – LL_SRDAMR_GRP1_PERIPH_LPTIM1
  – LL_SRDAMR_GRP1_PERIPH_LPTIM3
  – LL_SRDAMR_GRP1_PERIPH_LPTIM4
  – LL_SRDAMR_GRP1_PERIPH_OPAMP
  – LL_SRDAMR_GRP1_PERIPH_COMP
  – LL_SRDAMR_GRP1_PERIPH_VREF
  – LL_SRDAMR_GRP1_PERIPH_RTCAPB
  – LL_SRDAMR_GRP1_PERIPH_ADC4
  – LL_SRDAMR_GRP1_PERIPH_LPGPIO1
  – LL_SRDAMR_GRP1_PERIPH_DAC1
  – LL_SRDAMR_GRP1_PERIPH_LPDMA1
  – LL_SRDAMR_GRP1_PERIPH_ADF1
  – LL_SRDAMR_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- SRDAMR SPI3AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR LPUART1AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR I2C3AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR LPTIM1AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR LPTIM3AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR LPTIM4AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR OPAMPAMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR COMPAMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR VREFAMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR VREFRST LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR RTCAPBAMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR ADC4AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR LPGPIO1AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR DAC1AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR LPDMA1AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR ADF1AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
- SRDAMR SRAM4AMEN LL_SRDAMR_GRP1_EnableAutonomousClock
-

LL_SRDAMR_GRP1_IsEnabledAutonomousClock

Function name

__STATIC_INLINE uint32_t LL_SRDAMR_GRP1_IsEnabledAutonomousClock (uint32_t Periphs)

Function description

Check if SRDAMR peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_SRDAMR_GRP1_PERIPH_ALL
  - LL_SRDAMR_GRP1_PERIPH_SPI3
  - LL_SRDAMR_GRP1_PERIPH_LPUART1
  - LL_SRDAMR_GRP1_PERIPH_I2C3
  - LL_SRDAMR_GRP1_PERIPH_LPTIM1
  - LL_SRDAMR_GRP1_PERIPH_LPTIM3
  - LL_SRDAMR_GRP1_PERIPH_LPTIM4
  - LL_SRDAMR_GRP1_PERIPH_OPAMP
  - LL_SRDAMR_GRP1_PERIPH_COMP
  - LL_SRDAMR_GRP1_PERIPH_VREF
  - LL_SRDAMR_GRP1_PERIPH_RTCAPB
  - LL_SRDAMR_GRP1_PERIPH_ADC4
  - LL_SRDAMR_GRP1_PERIPH_LPGPIO1
  - LL_SRDAMR_GRP1_PERIPH_DAC1
  - LL_SRDAMR_GRP1_PERIPH_LPDMA1
  - LL_SRDAMR_GRP1_PERIPH_ADF1
  - LL_SRDAMR_GRP1_PERIPH_SRAM4

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- SRDAMR SPI3AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR LPUART1AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR I2C3AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR LPTIM1AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR LPTIM3AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR LPTIM4AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR OPAMPAMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR COMPAMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR VREFAMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR VREFRST LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR RTCAPBAMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR ADC4AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR LPGPIO1AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR DAC1AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR LPDMA1AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR ADF1AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
- SRDAMR SRAM4AMEN LL_SRDAMR_GRP1_IsEnabledAutonomousClock
-

LL_SRDAMR_GRP1_DisableAutonomousClock

Function name

__STATIC_INLINE void LL_SRDAMR_GRP1_DisableAutonomousClock (uint32_t Periphs)

Function description

Disable SRDAMR peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_SRDAMR_GRP1_PERIPH_ALL
  – LL_SRDAMR_GRP1_PERIPH_SPI3
  – LL_SRDAMR_GRP1_PERIPH_LPUART1
  – LL_SRDAMR_GRP1_PERIPH_I2C3
  – LL_SRDAMR_GRP1_PERIPH_LPTIM1
  – LL_SRDAMR_GRP1_PERIPH_LPTIM3
  – LL_SRDAMR_GRP1_PERIPH_LPTIM4
  – LL_SRDAMR_GRP1_PERIPH_OPAMP
  – LL_SRDAMR_GRP1_PERIPH_COMP
  – LL_SRDAMR_GRP1_PERIPH_VREF
  – LL_SRDAMR_GRP1_PERIPH_RTCAPB
  – LL_SRDAMR_GRP1_PERIPH_ADC4
  – LL_SRDAMR_GRP1_PERIPH_LPGPIO1
  – LL_SRDAMR_GRP1_PERIPH_DAC1
  – LL_SRDAMR_GRP1_PERIPH_LPDMA1
  – LL_SRDAMR_GRP1_PERIPH_ADF1
  – LL_SRDAMR_GRP1_PERIPH_SRAM4

Return values

- **None:**

Reference Manual to LL API cross reference:

- SRDAMR SPI3AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR LPUART1AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR I2C3AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR LPTIM1AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR LPTIM3AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR LPTIM4AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR OPAMPAMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR COMPAMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR VREFAMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR VREFRST LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR RTCAPBAMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR ADC4AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR LPGPIO1AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR DAC1AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR LPDMA1AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR ADF1AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
- SRDAMR SRAM4AMEN LL_SRDAMR_GRP1_DisableAutonomousClock
-

91.2 BUS Firmware driver defines

The following section lists the various define and macros of the module.

91.2.1 BUS

BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_GPDMA1

LL_AHB1_GRP1_PERIPH_CORDIC

LL_AHB1_GRP1_PERIPH_FMAC

LL_AHB1_GRP1_PERIPH_MDF1

LL_AHB1_GRP1_PERIPH_FLASH

LL_AHB1_GRP1_PERIPH_CRC

LL_AHB1_GRP1_PERIPH_JPEG

LL_AHB1_GRP1_PERIPH_TSC

LL_AHB1_GRP1_PERIPH_RAMCFG

LL_AHB1_GRP1_PERIPH_DMA2D

LL_AHB1_GRP1_PERIPH_GFXMMU

LL_AHB1_GRP1_PERIPH_GPU2D

LL_AHB1_GRP1_PERIPH_DCACHE2

LL_AHB1_GRP1_PERIPH_GTZC1

LL_AHB1_GRP1_PERIPH_BKPSRAM

LL_AHB1_GRP1_PERIPH_ICACHE1

LL_AHB1_GRP1_PERIPH_DCACHE1

LL_AHB1_GRP1_PERIPH_SRAM1

AHB2 GRP1 PERIPH

LL_AHB2_GRP1_PERIPH_ALL

LL_AHB2_GRP1_PERIPH_GPIOA

LL_AHB2_GRP1_PERIPH_GPIOB

LL_AHB2_GRP1_PERIPH_GPIOC

LL_AHB2_GRP1_PERIPH_GPIOD

LL_AHB2_GRP1_PERIPH_GPIOE

LL_AHB2_GRP1_PERIPH_GPIOF

LL_AHB2_GRP1_PERIPH_GPIOG

LL_AHB2_GRP1_PERIPH_GPIOH

LL_AHB2_GRP1_PERIPH_GPIOI

LL_AHB2_GRP1_PERIPH_GPIOJ

LL_AHB2_GRP1_PERIPH_ADC12

LL_AHB2_GRP1_PERIPH_DCMI_PSSI

LL_AHB2_GRP1_PERIPH_OTG_HS

LL_AHB2_GRP1_PERIPH_USBHS

LL_AHB2_GRP1_PERIPH_USBPHY

LL_AHB2_GRP1_PERIPH_AES

LL_AHB2_GRP1_PERIPH_HASH

LL_AHB2_GRP1_PERIPH_RNG

LL_AHB2_GRP1_PERIPH_PKA

LL_AHB2_GRP1_PERIPH_SAES

LL_AHB2_GRP1_PERIPH_OCTOSPIM

LL_AHB2_GRP1_PERIPH_OTFDEC1

LL_AHB2_GRP1_PERIPH_OTFDEC2

LL_AHB2_GRP1_PERIPH_SDMMC1

LL_AHB2_GRP1_PERIPH_SDMMC2

LL_AHB2_GRP1_PERIPH_SRAM2

LL_AHB2_GRP1_PERIPH_SRAM3

AHB2 GRP2 PERIPH

LL_AHB2_GRP2_PERIPH_ALL

LL_AHB2_GRP2_PERIPH_FSMC

LL_AHB2_GRP2_PERIPH_OCTOSPI1

LL_AHB2_GRP2_PERIPH_OCTOSPI2

LL_AHB2_GRP2_PERIPH_HSPI1

LL_AHB2_GRP2_PERIPH_SRAM6

LL_AHB2_GRP2_PERIPH_SRAM5

AHB3 GRP1 PERIPH

LL_AHB3_GRP1_PERIPH_ALL

LL_AHB3_GRP1_PERIPH_LPGPIO1

LL_AHB3_GRP1_PERIPH_PWR

LL_AHB3_GRP1_PERIPH_ADC4

LL_AHB3_GRP1_PERIPH_DAC1

LL_AHB3_GRP1_PERIPH_LPDMA1

LL_AHB3_GRP1_PERIPH_ADF1

LL_AHB3_GRP1_PERIPH_GTZC2

LL_AHB3_GRP1_PERIPH_SRAM4

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_TIM4

LL_APB1_GRP1_PERIPH_TIM5

LL_APB1_GRP1_PERIPH_TIM6

LL_APB1_GRP1_PERIPH_TIM7

LL_APB1_GRP1_PERIPH_WWDG

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_UART4

LL_APB1_GRP1_PERIPH_UART5

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_CRS

LL_APB1_GRP1_PERIPH_USART6

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_ALL

LL_APB1_GRP2_PERIPH_I2C4

LL_APB1_GRP2_PERIPH_LPTIM2

LL_APB1_GRP2_PERIPH_FDCAN1

LL_APB1_GRP2_PERIPH_UCPD1

LL_APB1_GRP2_PERIPH_I2C5

LL_APB1_GRP2_PERIPH_I2C6

APB2 GRP1 PERIPH

LL_APB2_GRP1_PERIPH_ALL

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_TIM8

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_TIM15

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_SAI1

LL_APB2_GRP1_PERIPH_SAI2

LL_APB2_GRP1_PERIPH_GFXTIM

LL_APB2_GRP1_PERIPH_LTDC

LL_APB2_GRP1_PERIPH_DSI

APB3 GRP1 PERIPH

LL_APB3_GRP1_PERIPH_ALL

LL_APB3_GRP1_PERIPH_SYSCFG

LL_APB3_GRP1_PERIPH_SPI3

LL_APB3_GRP1_PERIPH_LPUART1

LL_APB3_GRP1_PERIPH_I2C3

LL_APB3_GRP1_PERIPH_LPTIM1

LL_APB3_GRP1_PERIPH_LPTIM3

LL_APB3_GRP1_PERIPH_LPTIM4

LL_APB3_GRP1_PERIPH_OPAMP

LL_APB3_GRP1_PERIPH_COMP

LL_APB3_GRP1_PERIPH_VREF

LL_APB3_GRP1_PERIPH_RTCAPB

SRDAMR GRP1 PERIPH

LL_SRDAMR_GRP1_PERIPH_ALL

LL_SRDAMR_GRP1_PERIPH_SPI3

LL_SRDAMR_GRP1_PERIPH_LPUART1

LL_SRDAMR_GRP1_PERIPH_I2C3

LL_SRDAMR_GRP1_PERIPH_LPTIM1

LL_SRDAMR_GRP1_PERIPH_LPTIM3

LL_SRDAMR_GRP1_PERIPH_LPTIM4

LL_SRDAMR_GRP1_PERIPH_OPAMP

LL_SRDAMR_GRP1_PERIPH_COMP

LL_SRDAMR_GRP1_PERIPH_VREF

LL_SRDAMR_GRP1_PERIPH_RTCAPB

LL_SRDAMR_GRP1_PERIPH_ADC4

LL_SRDAMR_GRP1_PERIPH_LPGPIO1

LL_SRDAMR_GRP1_PERIPH_DAC1

LL_SRDAMR_GRP1_PERIPH_LPDMA1

LL_SRDAMR_GRP1_PERIPH_ADF1

LL_SRDAMR_GRP1_PERIPH_SRAM4

92 LL COMP Generic Driver

92.1 COMP Firmware driver registers structures

92.1.1 LL_COMP_InitTypeDef

LL_COMP_InitTypeDef is defined in the stm32u5xx_ll_comp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t InputPlus***
- ***uint32_t InputMinus***
- ***uint32_t InputHysteresis***
- ***uint32_t OutputPolarity***
- ***uint32_t OutputBlankingSource***

Field Documentation

- ***uint32_t LL_COMP_InitTypeDef::PowerMode***
  Set comparator operating mode to adjust power and speed. This parameter can be a value of ***COMP_LL_EC_POWERMODE*** This feature can be modified afterwards using unitary function **LL_COMP_SetPowerMode()**.
- ***uint32_t LL_COMP_InitTypeDef::InputPlus***
  Set comparator input plus (non-inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_PLUS*** This feature can be modified afterwards using unitary function **LL_COMP_SetInputPlus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputMinus***
  Set comparator input minus (inverting input). This parameter can be a value of ***COMP_LL_EC_INPUT_MINUS*** This feature can be modified afterwards using unitary function **LL_COMP_SetInputMinus()**.
- ***uint32_t LL_COMP_InitTypeDef::InputHysteresis***
  Set comparator hysteresis mode of the input minus. This parameter can be a value of ***COMP_LL_EC_INPUT_HYSTERESIS*** This feature can be modified afterwards using unitary function **LL_COMP_SetInputHysteresis()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputPolarity***
  Set comparator output polarity. This parameter can be a value of ***COMP_LL_EC_OUTPUT_POLARITY*** This feature can be modified afterwards using unitary function **LL_COMP_SetOutputPolarity()**.
- ***uint32_t LL_COMP_InitTypeDef::OutputBlankingSource***
  Set comparator blanking source. This parameter can be a value of ***COMP_LL_EC_OUTPUT_BLANKING_SOURCE*** This feature can be modified afterwards using unitary function **LL_COMP_SetOutputBlankingSource()**.

92.2 COMP Firmware driver API description

The following section lists the various functions of the COMP library.

92.2.1 Detailed description of functions

LL_COMP_SetCommonWindowMode

Function name

__STATIC_INLINE void LL_COMP_SetCommonWindowMode (COMP_Common_TypeDef * COMPxy_COMMON, uint32_t WindowMode)

Function description

Set window mode of a pair of comparators instances (2 consecutive COMP instances COMP<x> and COMP<x+1>).

Parameters

- **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())
- **WindowMode:** This parameter can be one of the following values:
  – LL_COMP_WINDOWMODE_DISABLE
  – LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON
  – LL_COMP_WINDOWMODE_COMP2_INPUT_PLUS_COMMON (1)

  (1) Parameter not available on all STM32U5 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR WINMODE LL_COMP_SetCommonWindowMode

LL_COMP_GetCommonWindowMode

Function name

__STATIC_INLINE uint32_t LL_COMP_GetCommonWindowMode (const COMP_Common_TypeDef * COMPxy_COMMON)

Function description

Get window mode of a pair of comparators instances (2 consecutive COMP instances COMP<x> and COMP<x+1>).

Parameters

- **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  – LL_COMP_WINDOWMODE_DISABLE
  – LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON
  – LL_COMP_WINDOWMODE_COMP2_INPUT_PLUS_COMMON (1)

  (1) Parameter not available on all STM32U5 devices

Reference Manual to LL API cross reference:

- CSR WINMODE LL_COMP_GetCommonWindowMode

LL_COMP_SetCommonWindowOutput

Function name

__STATIC_INLINE void LL_COMP_SetCommonWindowOutput (COMP_Common_TypeDef * COMPxy_COMMON, uint32_t WindowOutput)

Function description

Set window output of a pair of comparators instances (2 consecutive COMP instances COMP<x> and COMP<x+1>).

Parameters

- **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())
- **WindowOutput:** This parameter can be one of the following values:
  – LL_COMP_WINDOWOUTPUT_EACH_COMP
  – LL_COMP_WINDOWOUTPUT_COMP1 (1)
  – LL_COMP_WINDOWOUTPUT_COMP2 (1)
  – LL_COMP_WINDOWOUTPUT_BOTH

  (1) Parameter available on all STM32U5 devices, must be used with comparator common instance COMP12_COMMON

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR WINOUT LL_COMP_SetCommonWindowOutput

LL_COMP_GetCommonWindowOutput

Function name

__STATIC_INLINE uint32_t LL_COMP_GetCommonWindowOutput (const COMP_Common_TypeDef * COMPxy_COMMON)

Function description

Get window output of a pair of comparators instances (2 consecutive COMP instances COMP<x> and COMP<x+1>).

Parameters

- **COMPxy_COMMON:** Comparator common instance (can be set directly from CMSIS definition or by using helper macro __LL_COMP_COMMON_INSTANCE())

Return values

- **Returned:** value can be one of the following values:
  – LL_COMP_WINDOWOUTPUT_EACH_COMP
  – LL_COMP_WINDOWOUTPUT_COMP1 (1)
  – LL_COMP_WINDOWOUTPUT_COMP2 (1)
  – LL_COMP_WINDOWOUTPUT_BOTH

  (1) Parameter available on all STM32U5 devices, must be used with comparator common instance COMP12_COMMON

Reference Manual to LL API cross reference:

- CSR WINMODE LL_COMP_GetCommonWindowOutput

LL_COMP_SetPowerMode

Function name

__STATIC_INLINE void LL_COMP_SetPowerMode (COMP_TypeDef * COMPx, uint32_t PowerMode)

Function description

Set comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_COMP_POWERMODE_HIGHSPEED
  - LL_COMP_POWERMODE_MEDIUMSPEED
  - LL_COMP_POWERMODE_ULTRALOWPOWER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PWRMODE LL_COMP_SetPowerMode

LL_COMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_COMP_GetPowerMode (const COMP_TypeDef * COMPx)

Function description

Get comparator instance operating mode to adjust power and speed.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_POWERMODE_HIGHSPEED
  - LL_COMP_POWERMODE_MEDIUMSPEED
  - LL_COMP_POWERMODE_ULTRALOWPOWER

Reference Manual to LL API cross reference:

- CSR PWRMODE LL_COMP_GetPowerMode

LL_COMP_ConfigInputs

Function name

__STATIC_INLINE void LL_COMP_ConfigInputs (COMP_TypeDef * COMPx, uint32_t InputMinus, uint32_t InputPlus)

Function description

Set comparator inputs minus (inverting) and plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  – LL_COMP_INPUT_MINUS_1_4VREFINT
  – LL_COMP_INPUT_MINUS_1_2VREFINT
  – LL_COMP_INPUT_MINUS_3_4VREFINT
  – LL_COMP_INPUT_MINUS_VREFINT
  – LL_COMP_INPUT_MINUS_DAC1_CH1
  – LL_COMP_INPUT_MINUS_DAC1_CH2
  – LL_COMP_INPUT_MINUS_IO1
  – LL_COMP_INPUT_MINUS_IO2
- **InputPlus:** This parameter can be one of the following values:
  – LL_COMP_INPUT_PLUS_IO1
  – LL_COMP_INPUT_PLUS_IO2
  – LL_COMP_INPUT_PLUS_IO3

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 series, a voltage scaler is used when COMP input is based on VrefInt (VrefInt or subdivision of VrefInt): Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".

Reference Manual to LL API cross reference:

- CSR INMSEL LL_COMP_ConfigInputs
- CSR INPSEL LL_COMP_ConfigInputs
-

LL_COMP_SetInputPlus

Function name

__STATIC_INLINE void LL_COMP_SetInputPlus (COMP_TypeDef * COMPx, uint32_t InputPlus)

Function description

Set comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance
- **InputPlus:** This parameter can be one of the following values:
  – LL_COMP_INPUT_PLUS_IO1
  – LL_COMP_INPUT_PLUS_IO2
  – LL_COMP_INPUT_PLUS_IO3

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR INPSEL LL_COMP_SetInputPlus

LL_COMP_GetInputPlus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputPlus (const COMP_TypeDef * COMPx)

Function description

Get comparator input plus (non-inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  – LL_COMP_INPUT_PLUS_IO1
  – LL_COMP_INPUT_PLUS_IO2
  – LL_COMP_INPUT_PLUS_IO3

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR INPSEL LL_COMP_GetInputPlus

LL_COMP_SetInputMinus

Function name

__STATIC_INLINE void LL_COMP_SetInputMinus (COMP_TypeDef * COMPx, uint32_t InputMinus)

Function description

Set comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance
- **InputMinus:** This parameter can be one of the following values:
  – LL_COMP_INPUT_MINUS_1_4VREFINT
  – LL_COMP_INPUT_MINUS_1_2VREFINT
  – LL_COMP_INPUT_MINUS_3_4VREFINT
  – LL_COMP_INPUT_MINUS_VREFINT
  – LL_COMP_INPUT_MINUS_DAC1_CH1
  – LL_COMP_INPUT_MINUS_DAC1_CH2
  – LL_COMP_INPUT_MINUS_IO1
  – LL_COMP_INPUT_MINUS_IO2

Return values

- **None:**

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.
- On this STM32 series, a voltage scaler is used when COMP input is based on VrefInt (VrefInt or subdivision of VrefInt): Voltage scaler requires a delay for voltage stabilization. Refer to device datasheet, parameter "tSTART_SCALER".

Reference Manual to LL API cross reference:

- CSR INMSEL LL_COMP_SetInputMinus

LL_COMP_GetInputMinus

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputMinus (const COMP_TypeDef * COMPx)

Function description

Get comparator input minus (inverting).

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_INPUT_MINUS_1_4VREFINT
  - LL_COMP_INPUT_MINUS_1_2VREFINT
  - LL_COMP_INPUT_MINUS_3_4VREFINT
  - LL_COMP_INPUT_MINUS_VREFINT
  - LL_COMP_INPUT_MINUS_DAC1_CH1
  - LL_COMP_INPUT_MINUS_DAC1_CH2
  - LL_COMP_INPUT_MINUS_IO1
  - LL_COMP_INPUT_MINUS_IO2

Notes

- In case of comparator input selected to be connected to IO: GPIO pins are specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR INMSEL LL_COMP_GetInputMinus

LL_COMP_SetInputHysteresis

Function name

__STATIC_INLINE void LL_COMP_SetInputHysteresis (COMP_TypeDef * COMPx, uint32_t InputHysteresis)

Function description

Set comparator instance hysteresis mode of the input minus (inverting input).

Parameters

- **COMPx:** Comparator instance
- **InputHysteresis:** This parameter can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR HYST LL_COMP_SetInputHysteresis

LL_COMP_GetInputHysteresis

Function name

__STATIC_INLINE uint32_t LL_COMP_GetInputHysteresis (const COMP_TypeDef * COMPx)

Function description

Get comparator instance hysteresis mode of the minus (inverting) input.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_HYSTERESIS_NONE
  - LL_COMP_HYSTERESIS_LOW
  - LL_COMP_HYSTERESIS_MEDIUM
  - LL_COMP_HYSTERESIS_HIGH

Reference Manual to LL API cross reference:

- CSR HYST LL_COMP_GetInputHysteresis

LL_COMP_SetOutputPolarity

Function name

__STATIC_INLINE void LL_COMP_SetOutputPolarity (COMP_TypeDef * COMPx, uint32_t OutputPolarity)

Function description

Set comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance
- **OutputPolarity:** This parameter can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR POLARITY LL_COMP_SetOutputPolarity

LL_COMP_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputPolarity (const COMP_TypeDef * COMPx)

Function description

Get comparator instance output polarity.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  - LL_COMP_OUTPUTPOL_NONINVERTED
  - LL_COMP_OUTPUTPOL_INVERTED

Reference Manual to LL API cross reference:

- CSR POLARITY LL_COMP_GetOutputPolarity

LL_COMP_SetOutputBlankingSource

Function name

__STATIC_INLINE void LL_COMP_SetOutputBlankingSource (COMP_TypeDef * COMPx, uint32_t BlankingSource)

Function description

Set comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance
- **BlankingSource:** This parameter can be one of the following values:
  – LL_COMP_BLANKINGSRC_NONE
  – LL_COMP_BLANKINGSRC_TIM1_OC5
  – LL_COMP_BLANKINGSRC_TIM2_OC3
  – LL_COMP_BLANKINGSRC_TIM3_OC3
  – LL_COMP_BLANKINGSRC_TIM3_OC4
  – LL_COMP_BLANKINGSRC_TIM8_OC5
  – LL_COMP_BLANKINGSRC_TIM15_OC1

Return values

- **None:**

Notes

- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.
- Availability of parameters of blanking source from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CSR BLANKSEL LL_COMP_SetOutputBlankingSource

LL_COMP_GetOutputBlankingSource

Function name

__STATIC_INLINE uint32_t LL_COMP_GetOutputBlankingSource (const COMP_TypeDef * COMPx)

Function description

Get comparator instance blanking source.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  – LL_COMP_BLANKINGSRC_NONE
  – LL_COMP_BLANKINGSRC_TIM1_OC5
  – LL_COMP_BLANKINGSRC_TIM2_OC3
  – LL_COMP_BLANKINGSRC_TIM3_OC3
  – LL_COMP_BLANKINGSRC_TIM3_OC4
  – LL_COMP_BLANKINGSRC_TIM8_OC5
  – LL_COMP_BLANKINGSRC_TIM15_OC1

Notes

- Availability of parameters of blanking source from timer depends on timers availability on the selected device.
- Blanking source may be specific to each comparator instance. Refer to description of parameters or to reference manual.

Reference Manual to LL API cross reference:

- CSR BLANKSEL LL_COMP_GetOutputBlankingSource

LL_COMP_Enable

Function name

__STATIC_INLINE void LL_COMP_Enable (COMP_TypeDef * COMPx)

Function description

Enable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- After enable from off state, comparator requires a delay to reach reach propagation delay specification. Refer to device datasheet, parameter "tSTART".

Reference Manual to LL API cross reference:

- CSR EN LL_COMP_Enable

LL_COMP_Disable

Function name

__STATIC_INLINE void LL_COMP_Disable (COMP_TypeDef * COMPx)

Function description

Disable comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR EN LL_COMP_Disable

LL_COMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_COMP_IsEnabled (const COMP_TypeDef * COMPx)

Function description

Get comparator enable state (0: COMP is disabled, 1: COMP is enabled)

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR EN LL_COMP_IsEnabled

LL_COMP_Lock

Function name

__STATIC_INLINE void LL_COMP_Lock (COMP_TypeDef * COMPx)

Function description

Lock comparator instance.

Parameters

- **COMPx:** Comparator instance

Return values

- **None:**

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CSR LOCK LL_COMP_Lock

LL_COMP_IsLocked

Function name

__STATIC_INLINE uint32_t LL_COMP_IsLocked (const COMP_TypeDef * COMPx)

Function description

Get comparator lock state (0: COMP is unlocked, 1: COMP is locked).

Parameters

- **COMPx:** Comparator instance

Return values

- **State:** of bit (1 or 0).

Notes

- Once locked, comparator configuration can be accessed in read-only.
- The only way to unlock the comparator is a device hardware reset.

Reference Manual to LL API cross reference:

- CSR LOCK LL_COMP_IsLocked

LL_COMP_ReadOutputLevel

Function name

__STATIC_INLINE uint32_t LL_COMP_ReadOutputLevel (const COMP_TypeDef * COMPx)

Function description

Read comparator instance output level.

Parameters

- **COMPx:** Comparator instance

Return values

- **Returned:** value can be one of the following values:
  – LL_COMP_OUTPUT_LEVEL_LOW
  – LL_COMP_OUTPUT_LEVEL_HIGH

Notes

- The comparator output level depends on the selected polarity (Refer to function LL_COMP_SetOutputPolarity()). If the comparator polarity is not inverted: Comparator output is low when the input plus is at a lower voltage than the input minusComparator output is high when the input plus is at a higher voltage than the input minus If the comparator polarity is inverted:Comparator output is high when the input plus is at a lower voltage than the input minusComparator output is low when the input plus is at a higher voltage than the input minus

Reference Manual to LL API cross reference:

- CSR VALUE LL_COMP_ReadOutputLevel

LL_COMP_DeInit

Function name

ErrorStatus LL_COMP_DeInit (COMP_TypeDef * COMPx)

Function description

De-initialize registers of the selected COMP instance to their default reset values.

Parameters

- **COMPx:** COMP instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: COMP registers are de-initialized
  – ERROR: COMP registers are not de-initialized

Notes

- If comparator is locked, de-initialization by software is not possible. The only way to unlock the comparator is a device hardware reset.

LL_COMP_Init

Function name

ErrorStatus LL_COMP_Init (COMP_TypeDef * COMPx, const LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Initialize some features of COMP instance.

Parameters

- **COMPx:** COMP instance
- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: COMP registers are initialized
  – ERROR: COMP registers are not initialized

Notes

- This function configures features of the selected COMP instance. Some features are also available at scope COMP common instance (common to several COMP instances). Refer to functions having argument "COMPxy_COMMON" as parameter.

LL_COMP_StructInit

Function name

void LL_COMP_StructInit (LL_COMP_InitTypeDef * COMP_InitStruct)

Function description

Set each LL_COMP_InitTypeDef field to default value.

Parameters

- **COMP_InitStruct:** Pointer to a LL_COMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

92.3 COMP Firmware driver defines

The following section lists the various define and macros of the module.

92.3.1 COMP

COMP

Comparator common modes - Window mode

LL_COMP_WINDOWMODE_DISABLE

Window mode disable: Comparators 1 and 2 are independent

LL_COMP_WINDOWMODE_COMP1_INPUT_PLUS_COMMON

Window mode enable: Comparators instances pair COMP1 and COMP2 have their input plus connected together. The common input is COMP1 input plus (COMP2 input plus is no more accessible).

LL_COMP_WINDOWMODE_COMP2_INPUT_PLUS_COMMON

Window mode enable: if used from COMP1 or COMP2 instance, comparators instances pair COMP1 and COMP2 have their input plus connected together, the common input is COMP2 input plus (COMP1 input plus is no more accessible).

Comparator common modes - Window output

LL_COMP_WINDOWOUTPUT_EACH_COMP

Window output default mode: Comparators output are indicating each their own state. To know window mode state: each comparator output must be read, if "((COMPx exclusive or COMPy) == 1)" then monitored signal is within comparators window. The same way, if both comparators output are high, then monitored signal is below window.

LL_COMP_WINDOWOUTPUT_COMP1

Window output synthesized on COMP1 output: COMP1 output is no more indicating its own state, but global window mode state (logical high means monitored signal is within comparators window). Note: impacts only comparator output signal level (COMPx_OUT propagated to GPIO, EXTI lines, timers, ...), does not impact output digital state of comparator (COMPx_VALUE) always reflecting each comparator output state.

LL_COMP_WINDOWOUTPUT_COMP2

Window output synthesized on COMP2 output: COMP2 output is no more indicating its own state, but global window mode state (logical high means monitored signal is within comparators window). Note: impacts only comparator output signal level (COMPx_OUT propagated to GPIO, EXTI lines, timers, ...), does not impact output digital state of comparator (COMPx_VALUE) always reflecting each comparator output state.

LL_COMP_WINDOWOUTPUT_BOTH

Window output synthesized on both comparators output of pair of comparator selected (COMP1 and COMP2): both comparators outputs are no more indicating their own state, but global window mode state (logical high means monitored signal is within comparators window). This is a specific configuration (technically possible but not relevant from application point of view: 2 comparators output used for the same signal level), standard configuration for window mode is one of the settings above.

Definitions of COMP hardware constraints delays

LL_COMP_DELAY_STARTUP_US

Delay for COMP startup time

LL_COMP_DELAY_VOLTAGE_SCALER_STAB_US

Delay for COMP voltage scaler stabilization time

Comparator input - Hysteresis

LL_COMP_HYSTERESIS_NONE

No hysteresis

LL_COMP_HYSTERESIS_LOW

Hysteresis level low

LL_COMP_HYSTERESIS_MEDIUM

Hysteresis level medium

LL_COMP_HYSTERESIS_HIGH

Hysteresis level high

Comparator inputs - Input minus (input inverting) selection

LL_COMP_INPUT_MINUS_1_4VREFINT

Comparator input minus connected to 1/4 VrefInt

LL_COMP_INPUT_MINUS_1_2VREFINT

Comparator input minus connected to 1/2 VrefInt

LL_COMP_INPUT_MINUS_3_4VREFINT

Comparator input minus connected to 3/4 VrefInt

LL_COMP_INPUT_MINUS_VREFINT

Comparator input minus connected to VrefInt

LL_COMP_INPUT_MINUS_DAC1_CH1

Comparator input minus connected to DAC1 channel 1 (DAC_OUT1)

LL_COMP_INPUT_MINUS_DAC1_CH2

Comparator input minus connected to DAC1 channel 2 (DAC_OUT2)

LL_COMP_INPUT_MINUS_IO1

Comparator input minus connected to IO1 (pin PB1 for COMP1, pin PB3 for COMP2)

LL_COMP_INPUT_MINUS_IO2

Comparator input minus connected to IO2 (pin PC4 for COMP1, pin PB7 for COMP2)

Comparator inputs - Input plus (input non-inverting) selection

LL_COMP_INPUT_PLUS_IO1

Comparator input plus connected to IO1 (pin PC5 for COMP1, pin PB4 for COMP2)

LL_COMP_INPUT_PLUS_IO2

Comparator input plus connected to IO2 (pin PB2 for COMP1, pin PB6 for COMP2)

LL_COMP_INPUT_PLUS_IO3

Comparator input plus connected to IO3 (pin PA2 for COMP1)

Comparator output - Blanking source

LL_COMP_BLANKINGSRC_NONE

Comparator output without blanking

LL_COMP_BLANKINGSRC_TIM1_OC5

Comparator output blanking source TIM1 OC5 for COMP1 Instance

LL_COMP_BLANKINGSRC_TIM2_OC3

Comparator output blanking source TIM2 OC3 for COMP1 Instance

LL_COMP_BLANKINGSRC_TIM3_OC3

Comparator output blanking source TIM3 OC3 for COMP1 Instance

LL_COMP_BLANKINGSRC_TIM3_OC4

Comparator output blanking source TIM3 OC4 for COMP2 Instance

LL_COMP_BLANKINGSRC_TIM8_OC5

Comparator output blanking source TIM8 OC5 for COMP2 Instance

LL_COMP_BLANKINGSRC_TIM15_OC1

Comparator output blanking source TIM15 OC1 for COMP2 Instance

Comparator output - Output level

LL_COMP_OUTPUT_LEVEL_LOW

Comparator output level low (if the polarity is not inverted, otherwise to be complemented)

LL_COMP_OUTPUT_LEVEL_HIGH

Comparator output level high (if the polarity is not inverted, otherwise to be complemented)

Comparator output - Output polarity

LL_COMP_OUTPUTPOL_NONINVERTED

COMP output polarity is not inverted: comparator output is high when the plus (non-inverting) input is at a higher voltage than the minus (inverting) input

LL_COMP_OUTPUTPOL_INVERTED

COMP output polarity is inverted: comparator output is low when the plus (non-inverting) input is at a lower voltage than the minus (inverting) input

Comparator modes - Power mode

LL_COMP_POWERMODE_HIGHSPEED

COMP power mode to high speed

LL_COMP_POWERMODE_MEDIUMSPEED

COMP power mode to medium speed

LL_COMP_POWERMODE_ULTRALOWPOWER

COMP power mode to ultra-low power

COMP helper macro

__LL_COMP_COMMON_INSTANCE

Description:

- Helper macro to select the COMP common instance to which is belonging the selected COMP instance.

Parameters:

- __COMPx__: COMP instance

Return value:

- COMP: common instance or value "0" if there is no COMP common instance.

Notes:

- COMP common register instance can be used to set parameters common to several COMP instances. Refer to functions having argument "COMPxy_COMMON" as parameter.

Common write and read registers macro

LL_COMP_WriteReg

Description:

- Write a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_COMP_ReadReg

Description:

- Read a value in COMP register.

Parameters:

- __INSTANCE__: comparator instance
- __REG__: Register to be read

Return value:

- Register: value

93 LL CORDIC Generic Driver

93.1 CORDIC Firmware driver API description

The following section lists the various functions of the CORDIC library.

93.1.1 Detailed description of functions

LL_CORDIC_Config

Function name

__STATIC_INLINE void LL_CORDIC_Config (CORDIC_TypeDef * CORDICx, uint32_t Function, uint32_t Precision, uint32_t Scale, uint32_t NbWrite, uint32_t NbRead, uint32_t InSize, uint32_t OutSize)

Function description

Configure the CORDIC processing.

Parameters

- **CORDICx:** CORDIC instance
- **Function:** parameter can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7
- **NbWrite:** parameter can be one of the following values:
  - LL_CORDIC_NBWRITE_1
  - LL_CORDIC_NBWRITE_2
- **NbRead:** parameter can be one of the following values:
  - LL_CORDIC_NBREAD_1
  - LL_CORDIC_NBREAD_2
- **InSize:** parameter can be one of the following values:
  - LL_CORDIC_INSIZE_32BITS
  - LL_CORDIC_INSIZE_16BITS
- **OutSize:** parameter can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Notes

- This function set all parameters of CORDIC processing. These parameters can also be set individually using dedicated functions: LL_CORDIC_SetFunction()LL_CORDIC_SetPrecision()LL_CORDIC_SetScale()LL_CORDIC_SetNbWrite()LL_CORDIC_SetNbRead()LL_CORDIC_SetInSize()LL_CORDIC_SetOutSize()

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_Config
- CSR PRECISION LL_CORDIC_Config
- CSR SCALE LL_CORDIC_Config
- CSR NARGS LL_CORDIC_Config
- CSR NRES LL_CORDIC_Config
- CSR ARGSIZE LL_CORDIC_Config
- CSR RESIZE LL_CORDIC_Config

LL_CORDIC_SetFunction

Function name

__STATIC_INLINE void LL_CORDIC_SetFunction (CORDIC_TypeDef * CORDICx, uint32_t Function)

Function description

Configure function.

Parameters

- **CORDICx:** CORDIC Instance
- **Function:** parameter can be one of the following values:
  – LL_CORDIC_FUNCTION_COSINE
  – LL_CORDIC_FUNCTION_SINE
  – LL_CORDIC_FUNCTION_PHASE
  – LL_CORDIC_FUNCTION_MODULUS
  – LL_CORDIC_FUNCTION_ARCTANGENT
  – LL_CORDIC_FUNCTION_HCOSINE
  – LL_CORDIC_FUNCTION_HSINE
  – LL_CORDIC_FUNCTION_HARCTANGENT
  – LL_CORDIC_FUNCTION_NATURALLOG
  – LL_CORDIC_FUNCTION_SQUAREROOT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_SetFunction

LL_CORDIC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetFunction (const CORDIC_TypeDef * CORDICx)

Function description

Return function.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_FUNCTION_COSINE
  - LL_CORDIC_FUNCTION_SINE
  - LL_CORDIC_FUNCTION_PHASE
  - LL_CORDIC_FUNCTION_MODULUS
  - LL_CORDIC_FUNCTION_ARCTANGENT
  - LL_CORDIC_FUNCTION_HCOSINE
  - LL_CORDIC_FUNCTION_HSINE
  - LL_CORDIC_FUNCTION_HARCTANGENT
  - LL_CORDIC_FUNCTION_NATURALLOG
  - LL_CORDIC_FUNCTION_SQUAREROOT

Reference Manual to LL API cross reference:

- CSR FUNC LL_CORDIC_GetFunction

LL_CORDIC_SetPrecision

Function name

__STATIC_INLINE void LL_CORDIC_SetPrecision (CORDIC_TypeDef * CORDICx, uint32_t Precision)

Function description

Configure precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance
- **Precision:** parameter can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_SetPrecision

LL_CORDIC_GetPrecision

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetPrecision (const CORDIC_TypeDef * CORDICx)

Function description

Return precision in cycles number.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_PRECISION_1CYCLE
  - LL_CORDIC_PRECISION_2CYCLES
  - LL_CORDIC_PRECISION_3CYCLES
  - LL_CORDIC_PRECISION_4CYCLES
  - LL_CORDIC_PRECISION_5CYCLES
  - LL_CORDIC_PRECISION_6CYCLES
  - LL_CORDIC_PRECISION_7CYCLES
  - LL_CORDIC_PRECISION_8CYCLES
  - LL_CORDIC_PRECISION_9CYCLES
  - LL_CORDIC_PRECISION_10CYCLES
  - LL_CORDIC_PRECISION_11CYCLES
  - LL_CORDIC_PRECISION_12CYCLES
  - LL_CORDIC_PRECISION_13CYCLES
  - LL_CORDIC_PRECISION_14CYCLES
  - LL_CORDIC_PRECISION_15CYCLES

Reference Manual to LL API cross reference:

- CSR PRECISION LL_CORDIC_GetPrecision

LL_CORDIC_SetScale

Function name

__STATIC_INLINE void LL_CORDIC_SetScale (CORDIC_TypeDef * CORDICx, uint32_t Scale)

Function description

Configure scaling factor.

Parameters

- **CORDICx:** CORDIC Instance
- **Scale:** parameter can be one of the following values:
  - LL_CORDIC_SCALE_0
  - LL_CORDIC_SCALE_1
  - LL_CORDIC_SCALE_2
  - LL_CORDIC_SCALE_3
  - LL_CORDIC_SCALE_4
  - LL_CORDIC_SCALE_5
  - LL_CORDIC_SCALE_6
  - LL_CORDIC_SCALE_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_SetScale

LL_CORDIC_GetScale

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetScale (const CORDIC_TypeDef * CORDICx)

Function description

Return scaling factor.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_CORDIC_SCALE_0
  – LL_CORDIC_SCALE_1
  – LL_CORDIC_SCALE_2
  – LL_CORDIC_SCALE_3
  – LL_CORDIC_SCALE_4
  – LL_CORDIC_SCALE_5
  – LL_CORDIC_SCALE_6
  – LL_CORDIC_SCALE_7

Reference Manual to LL API cross reference:

- CSR SCALE LL_CORDIC_GetScale

LL_CORDIC_SetNbWrite

Function name

__STATIC_INLINE void LL_CORDIC_SetNbWrite (CORDIC_TypeDef * CORDICx, uint32_t NbWrite)

Function description

Configure number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbWrite:** parameter can be one of the following values:
  – LL_CORDIC_NBWRITE_1
  – LL_CORDIC_NBWRITE_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_SetNbWrite

LL_CORDIC_GetNbWrite

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbWrite (const CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit write expected for one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_CORDIC_NBWRITE_1
  – LL_CORDIC_NBWRITE_2

Reference Manual to LL API cross reference:

- CSR NARGS LL_CORDIC_GetNbWrite

LL_CORDIC_SetNbRead

Function name

__STATIC_INLINE void LL_CORDIC_SetNbRead (CORDIC_TypeDef * CORDICx, uint32_t NbRead)

Function description

Configure number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance
- **NbRead:** parameter can be one of the following values:
  – LL_CORDIC_NBREAD_1
  – LL_CORDIC_NBREAD_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_SetNbRead

LL_CORDIC_GetNbRead

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetNbRead (const CORDIC_TypeDef * CORDICx)

Function description

Return number of 32-bit read expected after one calculation.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_CORDIC_NBREAD_1
  – LL_CORDIC_NBREAD_2

Reference Manual to LL API cross reference:

- CSR NRES LL_CORDIC_GetNbRead

LL_CORDIC_SetInSize

Function name

__STATIC_INLINE void LL_CORDIC_SetInSize (CORDIC_TypeDef * CORDICx, uint32_t InSize)

Function description

Configure width of input data.

Parameters

- **CORDICx:** CORDIC Instance
- **InSize:** parameter can be one of the following values:
  – LL_CORDIC_INSIZE_32BITS
  – LL_CORDIC_INSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_SetInSize

LL_CORDIC_GetInSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetInSize (const CORDIC_TypeDef * CORDICx)

Function description

Return width of input data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_CORDIC_INSIZE_32BITS
  – LL_CORDIC_INSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR ARGSIZE LL_CORDIC_GetInSize

LL_CORDIC_SetOutSize

Function name

__STATIC_INLINE void LL_CORDIC_SetOutSize (CORDIC_TypeDef * CORDICx, uint32_t OutSize)

Function description

Configure width of output data.

Parameters

- **CORDICx:** CORDIC Instance
- **OutSize:** parameter can be one of the following values:
  – LL_CORDIC_OUTSIZE_32BITS
  – LL_CORDIC_OUTSIZE_16BITS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RESIZE LL_CORDIC_SetOutSize

LL_CORDIC_GetOutSize

Function name

__STATIC_INLINE uint32_t LL_CORDIC_GetOutSize (const CORDIC_TypeDef * CORDICx)

Function description

Return width of output data.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CORDIC_OUTSIZE_32BITS
  - LL_CORDIC_OUTSIZE_16BITS

Reference Manual to LL API cross reference:

- CSR RESIZE LL_CORDIC_GetOutSize

LL_CORDIC_EnableIT

Function name

__STATIC_INLINE void LL_CORDIC_EnableIT (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_EnableIT

LL_CORDIC_DisableIT

Function name

__STATIC_INLINE void LL_CORDIC_DisableIT (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC result ready interrupt.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_DisableIT

LL_CORDIC_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledIT (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready interrupt state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR IEN LL_CORDIC_IsEnabledIT

LL_CORDIC_EnableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_EnableDMAReq_RD

LL_CORDIC_DisableDMAReq_RD

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_RD (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA read channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_DisableDMAReq_RD

LL_CORDIC_IsEnabledDMAReq_RD

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_RD (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA read channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAREN LL_CORDIC_IsEnabledDMAReq_RD

LL_CORDIC_EnableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_EnableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Enable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_EnableDMAReq_WR

LL_CORDIC_DisableDMAReq_WR

Function name

__STATIC_INLINE void LL_CORDIC_DisableDMAReq_WR (CORDIC_TypeDef * CORDICx)

Function description

Disable CORDIC DMA write channel request.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_DisableDMAReq_WR

LL_CORDIC_IsEnabledDMAReq_WR

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsEnabledDMAReq_WR (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC DMA write channel request state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR DMAWEN LL_CORDIC_IsEnabledDMAReq_WR

LL_CORDIC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_CORDIC_DMA_GetRegAddr (const CORDIC_TypeDef * CORDICx, uint32_t Direction)

Function description

Get the CORDIC data register address used for DMA transfer.

Parameters

- **CORDICx:** CORDIC Instance
- **Direction:** parameter can be one of the following values:
  – LL_CORDIC_DMA_REG_DATA_IN
  – LL_CORDIC_DMA_REG_DATA_OUT

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_DMA_GetRegAddr
-
- WDATA ARG LL_CORDIC_DMA_GetRegAddr

LL_CORDIC_IsActiveFlag_RRDY

Function name

__STATIC_INLINE uint32_t LL_CORDIC_IsActiveFlag_RRDY (const CORDIC_TypeDef * CORDICx)

Function description

Check CORDIC result ready flag state.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR RRDY LL_CORDIC_IsActiveFlag_RRDY

LL_CORDIC_WriteData

Function name

__STATIC_INLINE void LL_CORDIC_WriteData (CORDIC_TypeDef * CORDICx, uint32_t InData)

Function description

Write 32-bit input data for the CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance
- **InData:** 0 .. 0xFFFFFFFF : 32-bit value to be provided as input data for CORDIC processing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA ARG LL_CORDIC_WriteData

LL_CORDIC_ReadData

Function name

__STATIC_INLINE uint32_t LL_CORDIC_ReadData (const CORDIC_TypeDef * CORDICx)

Function description

Return 32-bit output data of CORDIC processing.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **32-bit:** output data of CORDIC processing.

Reference Manual to LL API cross reference:

- RDATA RES LL_CORDIC_ReadData

LL_CORDIC_DeInit

Function name

ErrorStatus LL_CORDIC_DeInit (const CORDIC_TypeDef * CORDICx)

Function description

De-Initialize CORDIC peripheral registers to their default reset values.

Parameters

- **CORDICx:** CORDIC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CORDIC registers are de-initialized
  - ERROR: CORDIC registers are not de-initialized

93.2 CORDIC Firmware driver defines

The following section lists the various define and macros of the module.

93.2.1 CORDIC

CORDIC

DMA register data

LL_CORDIC_DMA_REG_DATA_IN

Get address of input data register

LL_CORDIC_DMA_REG_DATA_OUT

Get address of output data register

FUNCTION

LL_CORDIC_FUNCTION_COSINE

Cosine

LL_CORDIC_FUNCTION_SINE

Sine

LL_CORDIC_FUNCTION_PHASE

Phase

LL_CORDIC_FUNCTION_MODULUS

Modulus

LL_CORDIC_FUNCTION_ARCTANGENT

Arctangent

LL_CORDIC_FUNCTION_HCOSINE

Hyperbolic Cosine

LL_CORDIC_FUNCTION_HSINE

Hyperbolic Sine

LL_CORDIC_FUNCTION_HARCTANGENT

Hyperbolic Arctangent

LL_CORDIC_FUNCTION_NATURALLOG

Natural Logarithm

LL_CORDIC_FUNCTION_SQUAREROOT

Square Root

Get Flags Defines

LL_CORDIC_FLAG_RRDY

INSIZE

LL_CORDIC_INSIZE_32BITS

32 bits input data size (Q1.31 format)

LL_CORDIC_INSIZE_16BITS

16 bits input data size (Q1.15 format)

IT Defines

LL_CORDIC_IT_IEN

Result Ready interrupt enable

NBREAD

LL_CORDIC_NBREAD_1

One 32-bits read containing either only one 32-bit data output (Q1.31 format), or two 16-bit data output (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBREAD_2

Two 32-bit Data containing two 32-bits data output (Q1.31 format)

NBWRITE

LL_CORDIC_NBWRITE_1

One 32-bits write containing either only one 32-bit data input (Q1.31 format), or two 16-bit data input (Q1.15 format) packed in one 32 bits Data

LL_CORDIC_NBWRITE_2

Two 32-bit write containing two 32-bits data input (Q1.31 format)

OUTSIZE

LL_CORDIC_OUTSIZE_32BITS

32 bits output data size (Q1.31 format)

LL_CORDIC_OUTSIZE_16BITS

16 bits output data size (Q1.15 format)

PRECISION

LL_CORDIC_PRECISION_1CYCLE

LL_CORDIC_PRECISION_2CYCLES

LL_CORDIC_PRECISION_3CYCLES

LL_CORDIC_PRECISION_4CYCLES

LL_CORDIC_PRECISION_5CYCLES

LL_CORDIC_PRECISION_6CYCLES

LL_CORDIC_PRECISION_7CYCLES

LL_CORDIC_PRECISION_8CYCLES

LL_CORDIC_PRECISION_9CYCLES

LL_CORDIC_PRECISION_10CYCLES

LL_CORDIC_PRECISION_11CYCLES

LL_CORDIC_PRECISION_12CYCLES

LL_CORDIC_PRECISION_13CYCLES

LL_CORDIC_PRECISION_14CYCLES

LL_CORDIC_PRECISION_15CYCLES

SCALE

LL_CORDIC_SCALE_0

LL_CORDIC_SCALE_1

LL_CORDIC_SCALE_2

LL_CORDIC_SCALE_3

LL_CORDIC_SCALE_4

LL_CORDIC_SCALE_5

LL_CORDIC_SCALE_6

LL_CORDIC_SCALE_7

Common Write and read registers Macros

LL_CORDIC_WriteReg

Description:

- Write a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CORDIC_ReadReg

Description:

- Read a value in CORDIC register.

Parameters:

- __INSTANCE__: CORDIC Instance
- __REG__: Register to be read

Return value:

- Register: value

94 LL CORTEX Generic Driver

94.1 CORTEX Firmware driver API description

The following section lists the various functions of the CORTEX library.

94.1.1 Detailed description of functions

LL_SYSTICK_IsActiveCounterFlag

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsActiveCounterFlag (void)

Function description

This function checks if the Systick counter flag is active or not.

Return values

- **State:** of bit (1 or 0).

Notes

- It can be used in timeout function on application side.

Reference Manual to LL API cross reference:

- STK_CTRL COUNTFLAG LL_SYSTICK_IsActiveCounterFlag

LL_SYSTICK_SetClkSource

Function name

__STATIC_INLINE void LL_SYSTICK_SetClkSource (uint32_t Source)

Function description

Configures the SysTick clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_SYSTICK_CLKSOURCE_EXTERNAL
  – LL_SYSTICK_CLKSOURCE_HCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_SetClkSource

LL_SYSTICK_GetClkSource

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_GetClkSource (void)

Function description

Get the SysTick clock source.

Return values

- **Returned:** value can be one of the following values:
  – LL_SYSTICK_CLKSOURCE_EXTERNAL
  – LL_SYSTICK_CLKSOURCE_HCLK

Reference Manual to LL API cross reference:

- STK_CTRL CLKSOURCE LL_SYSTICK_GetClkSource

LL_SYSTICK_EnableIT

Function name

__STATIC_INLINE void LL_SYSTICK_EnableIT (void)

Function description

Enable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_EnableIT

LL_SYSTICK_DisableIT

Function name

__STATIC_INLINE void LL_SYSTICK_DisableIT (void)

Function description

Disable SysTick exception request.

Return values

- **None:**

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_DisableIT

LL_SYSTICK_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSTICK_IsEnabledIT (void)

Function description

Checks if the SYSTICK interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- STK_CTRL TICKINT LL_SYSTICK_IsEnabledIT

LL_LPM_EnableSleep

Function name

__STATIC_INLINE void LL_LPM_EnableSleep (void)

Function description

Processor uses sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableSleep

LL_LPM_EnableDeepSleep

Function name

__STATIC_INLINE void LL_LPM_EnableDeepSleep (void)

Function description

Processor uses deep sleep as its low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPDEEP LL_LPM_EnableDeepSleep

LL_LPM_EnableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_EnableSleepOnExit (void)

Function description

Configures sleep-on-exit when returning from Handler mode to Thread mode.

Return values

- **None:**

Notes

- Setting this bit to 1 enables an interrupt-driven application to avoid returning to an empty main application.

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_EnableSleepOnExit

LL_LPM_DisableSleepOnExit

Function name

__STATIC_INLINE void LL_LPM_DisableSleepOnExit (void)

Function description

Do not sleep when returning to Thread mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SLEEPONEXIT LL_LPM_DisableSleepOnExit

LL_LPM_EnableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_EnableEventOnPend (void)

Function description

Enabled events and all interrupts, including disabled interrupts, can wakeup the processor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_EnableEventOnPend

LL_LPM_DisableEventOnPend

Function name

__STATIC_INLINE void LL_LPM_DisableEventOnPend (void)

Function description

Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SCR SEVEONPEND LL_LPM_DisableEventOnPend

LL_HANDLER_EnableFault

Function name

__STATIC_INLINE void LL_HANDLER_EnableFault (uint32_t Fault)

Function description

Enable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  – LL_HANDLER_FAULT_USG
  – LL_HANDLER_FAULT_BUS
  – LL_HANDLER_FAULT_MEM
  – LL_HANDLER_FAULT_SECURE (*)

  (*) value applicable in secure when the system implements the security.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR USGFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR BUSFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR MEMFAULTENA LL_HANDLER_EnableFault
- SCB_SHCSR SECUREFAULTENA LL_HANDLER_EnableFault

LL_HANDLER_DisableFault

Function name

__STATIC_INLINE void LL_HANDLER_DisableFault (uint32_t Fault)

Function description

Disable a fault in System handler control register (SHCSR)

Parameters

- **Fault:** This parameter can be a combination of the following values:
  – LL_HANDLER_FAULT_USG
  – LL_HANDLER_FAULT_BUS
  – LL_HANDLER_FAULT_MEM
  – LL_HANDLER_FAULT_SECURE (*)

  (*) value applicable in secure when the system implements the security.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCB_SHCSR USGFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR BUSFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR MEMFAULTENA LL_HANDLER_DisableFault
- SCB_SHCSR SECUREFAULTENA LL_HANDLER_DisableFault

LL_CPUID_GetImplementer

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetImplementer (void)

Function description

Get Implementer code.

Return values

- **Value:** should be equal to 0x41 for ARM

Reference Manual to LL API cross reference:

- SCB_CPUID IMPLEMENTER LL_CPUID_GetImplementer

LL_CPUID_GetVariant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetVariant (void)

Function description

Get Variant number (The r value in the rnpn product revision identifier)

Return values

- **Value:** between 0 and 255 (0x0: revision 0)

Reference Manual to LL API cross reference:

- SCB_CPUID VARIANT LL_CPUID_GetVariant

LL_CPUID_GetArchitecture

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetArchitecture (void)

Function description

Get Architecture version.

Return values

- **Value:** should be equal to 0xF for Cortex-M33 ("ARMv8-M with Main Extension")

Reference Manual to LL API cross reference:

- SCB_CPUID ARCHITECTURE LL_CPUID_GetArchitecture

LL_CPUID_GetParNo

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetParNo (void)

Function description

Get Part number.

Return values

- **Value:** should be equal to 0xD21 for Cortex-M33

Reference Manual to LL API cross reference:

- SCB_CPUID PARTNO LL_CPUID_GetParNo

LL_CPUID_GetRevision

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetRevision (void)

Function description

Get Revision number (The p value in the rnpn product revision identifier, indicates patch release)

Return values

- **Value:** between 0 and 255 (0x1: patch 1)

Reference Manual to LL API cross reference:

- SCB_CPUID REVISION LL_CPUID_GetRevision

LL_MPU_Enable

Function name

__STATIC_INLINE void LL_MPU_Enable (uint32_t MPU_Control)

Function description

Enable MPU with input options.

Parameters

- **MPU_Control:** This parameter can be one of the following values:
  - LL_MPU_CTRL_HFNMI_PRIVDEF_NONE
  - LL_MPU_CTRL_HARDFAULT_NMI
  - LL_MPU_CTRL_PRIVILEGED_DEFAULT
  - LL_MPU_CTRL_HFNMI_PRIVDEF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Enable_NS

Function name

__STATIC_INLINE void LL_MPU_Enable_NS (uint32_t MPU_Control)

Function description

Enable non-secure MPU with input options.

Parameters

- **MPU_Control:** This parameter can be one of the following values:
  - LL_MPU_CTRL_HFNMI_PRIVDEF_NONE
  - LL_MPU_CTRL_HARDFAULT_NMI
  - LL_MPU_CTRL_PRIVILEGED_DEFAULT
  - LL_MPU_CTRL_HFNMI_PRIVDEF

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Enable

LL_MPU_Disable

Function name

__STATIC_INLINE void LL_MPU_Disable (void)

Function description

Disable MPU.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Disable

LL_MPU_Disable_NS

Function name

__STATIC_INLINE void LL_MPU_Disable_NS (void)

Function description

Disable the non-secure MPU.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_Disable_NS

LL_MPU_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled (void)

Function description

Check if MPU is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_IsEnabled

LL_MPU_IsEnabled_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled_NS (void)

Function description

Check if non-secure MPU is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MPU_CTRL ENABLE LL_MPU_IsEnabled_NS

LL_MPU_EnableRegion

Function name

__STATIC_INLINE void LL_MPU_EnableRegion (uint32_t Region)

Function description

Enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RLAR ENABLE LL_MPU_EnableRegion

LL_MPU_IsEnabled_Region

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled_Region (uint32_t Region)

Function description

Check if MPU region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **State:** of bit (1 or 0).

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR ENABLE LL_MPU_IsEnabled_Region

LL_MPU_EnableRegion_NS

Function name

__STATIC_INLINE void LL_MPU_EnableRegion_NS (uint32_t Region)

Function description

Enable a non-secure MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RLAR ENABLE LL_MPU_EnableRegion_NS

LL_MPU_IsEnabled_Region_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_IsEnabled_Region_NS (uint32_t Region)

Function description

Check if non-secure MPU region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7

Return values

- **State:** of bit (1 or 0).

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR ENABLE LL_MPU_IsEnabled_Region_NS

LL_MPU_DisableRegion

Function name

__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)

Function description

Disable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion
- MPU_RLAR ENABLE LL_MPU_DisableRegion

LL_MPU_DisableRegion_NS

Function name

__STATIC_INLINE void LL_MPU_DisableRegion_NS (uint32_t Region)

Function description

Disable a non-secure MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion_NS
- MPU_RLAR ENABLE LL_MPU_DisableRegion_NS
-

LL_MPU_ConfigRegion

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t Attributes, uint32_t AttrIndx, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure and enable a MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO
- **AttrIndx:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion
- MPU_RBAR ADDR LL_MPU_ConfigRegion
- MPU_RLAR ADDR LL_MPU_ConfigRegion
- MPU_RBAR XN LL_MPU_ConfigRegion
- MPU_RBAR AP LL_MPU_ConfigRegion
- MPU_RBAR SH LL_MPU_ConfigRegion
- MPU_RLAR EN LL_MPU_ConfigRegion
- MPU_RLAR AttrIndx LL_MPU_ConfigRegion
-

LL_MPU_ConfigRegion_NS

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion_NS (uint32_t Region, uint32_t Attributes, uint32_t AttrIndx, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure and enable a non-secure MPU region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO
- **AttrIndx:** This parameter can be one of the following values:
  - LL_MPU_ATTRIBUTES_NUMBER0
  - LL_MPU_ATTRIBUTES_NUMBER1
  - LL_MPU_ATTRIBUTES_NUMBER2
  - LL_MPU_ATTRIBUTES_NUMBER3
  - LL_MPU_ATTRIBUTES_NUMBER4
  - LL_MPU_ATTRIBUTES_NUMBER5
  - LL_MPU_ATTRIBUTES_NUMBER6
  - LL_MPU_ATTRIBUTES_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion_NS
- MPU_RBAR ADDR LL_MPU_ConfigRegion_NS
- MPU_RLAR ADDR LL_MPU_ConfigRegion_NS
- MPU_RBAR XN LL_MPU_ConfigRegion_NS
- MPU_RBAR AP LL_MPU_ConfigRegion_NS
- MPU_RBAR SH LL_MPU_ConfigRegion_NS
- MPU_RLAR EN LL_MPU_ConfigRegion_NS
- MPU_RLAR AttrIndx LL_MPU_ConfigRegion_NS
-

LL_MPU_ConfigRegionAddress

Function name

__STATIC_INLINE void LL_MPU_ConfigRegionAddress (uint32_t Region, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure a MPU region address range.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegionAddress
- MPU_RBAR ADDR LL_MPU_ConfigRegionAddress
- MPU_RLAR ADDR LL_MPU_ConfigRegionAddress
-

LL_MPU_ConfigRegionAddress_NS

Function name

__STATIC_INLINE void LL_MPU_ConfigRegionAddress_NS (uint32_t Region, uint32_t BaseAddress, uint32_t LimitAddress)

Function description

Configure a non-secure MPU region address range.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7
- **BaseAddress:** Value of region base address
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegionAddress_NS
- MPU_RBAR ADDR LL_MPU_ConfigRegionAddress_NS
- MPU_RLAR ADDR LL_MPU_ConfigRegionAddress_NS
-

LL_MPU_ConfigAttributes

Function name

__STATIC_INLINE void LL_MPU_ConfigAttributes (uint32_t AttIndex, uint32_t Attributes)

Function description

Configure a MPU attributes index.

Parameters

- **AttIndex:** This parameter can be one of the following values:
  – LL_MPU_ATTRIBUTES_NUMBER0
  – LL_MPU_ATTRIBUTES_NUMBER1
  – LL_MPU_ATTRIBUTES_NUMBER2
  – LL_MPU_ATTRIBUTES_NUMBER3
  – LL_MPU_ATTRIBUTES_NUMBER4
  – LL_MPU_ATTRIBUTES_NUMBER5
  – LL_MPU_ATTRIBUTES_NUMBER6
  – LL_MPU_ATTRIBUTES_NUMBER7
- **Attributes:** This parameter can be a combination of CORTEX LL MPU Attributes

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_MAIR0 Attribute LL_MPU_ConfigAttributes
- MPU_MAIR1 Attribute LL_MPU_ConfigAttributes
-

LL_MPU_ConfigAttributes_NS

Function name

__STATIC_INLINE void LL_MPU_ConfigAttributes_NS (uint32_t AttIndex, uint32_t Attributes)

Function description

Configure a non-secure MPU attributes index.

Parameters

- **AttIndex:** This parameter can be one of the following values:
  – LL_MPU_ATTRIBUTES_NUMBER0
  – LL_MPU_ATTRIBUTES_NUMBER1
  – LL_MPU_ATTRIBUTES_NUMBER2
  – LL_MPU_ATTRIBUTES_NUMBER3
  – LL_MPU_ATTRIBUTES_NUMBER4
  – LL_MPU_ATTRIBUTES_NUMBER5
  – LL_MPU_ATTRIBUTES_NUMBER6
  – LL_MPU_ATTRIBUTES_NUMBER7
- **Attributes:** This parameter can be a combination of CORTEX LL MPU Attributes

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_MAIR0 Attribute LL_MPU_ConfigAttributes_NS
- MPU_MAIR1 Attribute LL_MPU_ConfigAttributes_NS
-

LL_MPU_SetRegionLimitAddress

Function name

__STATIC_INLINE void LL_MPU_SetRegionLimitAddress (uint32_t Region, uint32_t LimitAddress)

Function description

Configure a MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionLimitAddress
- MPU_RLAR ADDR LL_MPU_SetRegionLimitAddress
-

LL_MPU_GetRegionLimitAddress

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionLimitAddress (uint32_t Region)

Function description

Get a MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionLimitAddress
-

LL_MPU_SetRegionBaseAddress

Function name

__STATIC_INLINE void LL_MPU_SetRegionBaseAddress (uint32_t Region, uint32_t BaseAddress)

Function description

Configure a MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **BaseAddress:** Value of region base address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionBaseAddress
- MPU_RBAR ADDR LL_MPU_SetRegionBaseAddress
-

LL_MPU_GetRegionBaseAddress

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionBaseAddress (uint32_t Region)

Function description

Get a MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionBaseAddress
-

LL_MPU_SetRegionAccess

Function name

__STATIC_INLINE void LL_MPU_SetRegionAccess (uint32_t Region, uint32_t Attributes)

Function description

Configure a MPU region access attributes and enable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionAccess
- MPU_RBAR XN LL_MPU_SetRegionAccess
- MPU_RBAR AP LL_MPU_SetRegionAccess
- MPU_RBAR SH LL_MPU_SetRegionAccess
-

LL_MPU_GetRegionAccess

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionAccess (uint32_t Region)

Function description

Get a MPU region access attributes.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionAccess
- MPU_RBAR XN LL_MPU_GetRegionAccess
- MPU_RBAR AP LL_MPU_GetRegionAccess
- MPU_RBAR SH LL_MPU_GetRegionAccess
-

LL_MPU_SetRegionLimitAddress_NS

Function name

__STATIC_INLINE void LL_MPU_SetRegionLimitAddress_NS (uint32_t Region, uint32_t LimitAddress)

Function description

Configure a non-secure MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7
- **LimitAddress:** Value of region limit address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionLimitAddress_NS
- MPU_RLAR ADDR LL_MPU_SetRegionLimitAddress_NS
-

LL_MPU_GetRegionLimitAddress_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionLimitAddress_NS (uint32_t Region)

Function description

Get a non-secure MPU region limit address.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_MPU_REGION_NUMBER0
  – LL_MPU_REGION_NUMBER1
  – LL_MPU_REGION_NUMBER2
  – LL_MPU_REGION_NUMBER3
  – LL_MPU_REGION_NUMBER4
  – LL_MPU_REGION_NUMBER5
  – LL_MPU_REGION_NUMBER6
  – LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionLimitAddress_NS
-

LL_MPU_SetRegionBaseAddress_NS

Function name

__STATIC_INLINE void LL_MPU_SetRegionBaseAddress_NS (uint32_t Region, uint32_t BaseAddress)

Function description

Configure a non-secure MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **BaseAddress:** Value of region base address

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionBaseAddress_NS
- MPU_RBAR ADDR LL_MPU_SetRegionBaseAddress_NS
-

LL_MPU_GetRegionBaseAddress_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionBaseAddress_NS (uint32_t Region)

Function description

Get a non-secure MPU region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionBaseAddress_NS
-

LL_MPU_SetRegionAccess_NS

Function name

__STATIC_INLINE void LL_MPU_SetRegionAccess_NS (uint32_t Region, uint32_t Attributes)

Function description

Configure a non-secure MPU region access attributes and enable a region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_NOT_SHAREABLE or LL_MPU_ACCESS_OUTER_SHAREABLE or LL_MPU_ACCESS_INNER_SHAREABLE
  - LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_ALL_RW or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_ALL_RO

Return values

- **None:**

Notes

- cortex-M33 supports 8 secure and 8 non secure regions.

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_SetRegionAccess_NS
- MPU_RBAR XN LL_MPU_SetRegionAccess_NS
- MPU_RBAR AP LL_MPU_SetRegionAccess_NS
- MPU_RBAR SH LL_MPU_SetRegionAccess_NS
-

LL_MPU_GetRegionAccess_NS

Function name

__STATIC_INLINE uint32_t LL_MPU_GetRegionAccess_NS (uint32_t Region)

Function description

Get a non-secure MPU region access attributes.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_MPU_REGION_NUMBER0
  - LL_MPU_REGION_NUMBER1
  - LL_MPU_REGION_NUMBER2
  - LL_MPU_REGION_NUMBER3
  - LL_MPU_REGION_NUMBER4
  - LL_MPU_REGION_NUMBER5
  - LL_MPU_REGION_NUMBER6
  - LL_MPU_REGION_NUMBER7

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_GetRegionAccess_NS
- MPU_RBAR XN LL_MPU_GetRegionAccess_NS
- MPU_RBAR AP LL_MPU_GetRegionAccess_NS
- MPU_RBAR SH LL_MPU_GetRegionAccess_NS
-

94.2 CORTEX Firmware driver defines

The following section lists the various define and macros of the module.

94.2.1 CORTEX

CORTEX

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_EXTERNAL

External clock source selected as SysTick clock source

LL_SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8

Handler Fault type

LL_HANDLER_FAULT_USG

Usage fault

LL_HANDLER_FAULT_BUS

Bus fault

LL_HANDLER_FAULT_MEM

Memory management fault

LL_HANDLER_FAULT_SECURE

Secure fault

CORTEX LL MPU Region Access Attributes

MPU_ACCESS_MSK

CORTEX LL MPU Instruction Access Shareable

LL_MPU_ACCESS_NOT_SHAREABLE

LL_MPU_ACCESS_OUTER_SHAREABLE

LL_MPU_ACCESS_INNER_SHAREABLE

CORTEX LL MPU Attributes

LL_MPU_DEVICE_NGNRNE

LL_MPU_DEVICE_NGNRE

LL_MPU_DEVICE_NGRE

LL_MPU_DEVICE_GRE

LL_MPU_WRITE_THROUGH

LL_MPU_NOT_CACHEABLE

LL_MPU_WRITE_BACK

LL_MPU_TRANSIENT

LL_MPU_NON_TRANSIENT

LL_MPU_NO_ALLOCATE

LL_MPU_W_ALLOCATE

LL_MPU_R_ALLOCATE

LL_MPU_RW_ALLOCATE

CORTEX LL MPU Memory Attributes Index

LL_MPU_ATTRIBUTES_NUMBER0

LL_MPU_ATTRIBUTES_NUMBER1

LL_MPU_ATTRIBUTES_NUMBER2

LL_MPU_ATTRIBUTES_NUMBER3

LL_MPU_ATTRIBUTES_NUMBER4

LL_MPU_ATTRIBUTES_NUMBER5

LL_MPU_ATTRIBUTES_NUMBER6

LL_MPU_ATTRIBUTES_NUMBER7

CORTEX LL MPU HFNMI and PRIVILEGED Access control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE

LL_MPU_CTRL_HARDFAULT_NMI

LL_MPU_CTRL_PRIVILEGED_DEFAULT

LL_MPU_CTRL_HFNMI_PRIVDEF

CORTEX LL MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE

LL_MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX LL MPU Region Enable

LL_MPU_REGION_ENABLE

LL_MPU_REGION_DISABLE

CORTEX LL MPU Region Index

LL_MPU_REGION_NUMBER0

LL_MPU_REGION_NUMBER1

LL_MPU_REGION_NUMBER2

LL_MPU_REGION_NUMBER3

LL_MPU_REGION_NUMBER4

LL_MPU_REGION_NUMBER5

LL_MPU_REGION_NUMBER6

LL_MPU_REGION_NUMBER7

CORTEX LL MPU Region Permission Attributes

LL_MPU_REGION_PRIV_RW

LL_MPU_REGION_ALL_RW

LL_MPU_REGION_PRIV_RO

LL_MPU_REGION_ALL_RO

95 LL CRC Generic Driver

95.1 CRC Firmware driver API description

The following section lists the various functions of the CRC library.

95.1.1 Detailed description of functions

LL_CRC_ResetCRCCalculationUnit

Function name

__STATIC_INLINE void LL_CRC_ResetCRCCalculationUnit (CRC_TypeDef * CRCx)

Function description

Reset the CRC calculation unit.

Parameters

- **CRCx:** CRC Instance

Return values

- **None:**

Notes

- If Programmable Initial CRC value feature is available, also set the Data Register to the value stored in the CRC_INIT register, otherwise, reset Data Register to its default value.

Reference Manual to LL API cross reference:

- CR RESET LL_CRC_ResetCRCCalculationUnit

LL_CRC_SetPolynomialSize

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialSize (CRC_TypeDef * CRCx, uint32_t PolySize)

Function description

Configure size of the polynomial.

Parameters

- **CRCx:** CRC Instance
- **PolySize:** This parameter can be one of the following values:
  – LL_CRC_POLYLENGTH_32B
  – LL_CRC_POLYLENGTH_16B
  – LL_CRC_POLYLENGTH_8B
  – LL_CRC_POLYLENGTH_7B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_SetPolynomialSize

LL_CRC_GetPolynomialSize

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialSize (const CRC_TypeDef * CRCx)

Function description

Return size of the polynomial.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_POLYLENGTH_32B
  - LL_CRC_POLYLENGTH_16B
  - LL_CRC_POLYLENGTH_8B
  - LL_CRC_POLYLENGTH_7B

Reference Manual to LL API cross reference:

- CR POLYSIZE LL_CRC_GetPolynomialSize

LL_CRC_SetInputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetInputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the input data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_SetInputDataReverseMode

LL_CRC_GetInputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInputDataReverseMode (const CRC_TypeDef * CRCx)

Function description

Return type of reversal for input data bit order.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_CRC_INDATA_REVERSE_NONE
  - LL_CRC_INDATA_REVERSE_BYTE
  - LL_CRC_INDATA_REVERSE_HALFWORD
  - LL_CRC_INDATA_REVERSE_WORD

Reference Manual to LL API cross reference:

- CR REV_IN LL_CRC_GetInputDataReverseMode

LL_CRC_SetOutputDataReverseMode

Function name

__STATIC_INLINE void LL_CRC_SetOutputDataReverseMode (CRC_TypeDef * CRCx, uint32_t ReverseMode)

Function description

Configure the reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance
- **ReverseMode:** This parameter can be one of the following values:
  – LL_CRC_OUTDATA_REVERSE_NONE
  – LL_CRC_OUTDATA_REVERSE_BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_SetOutputDataReverseMode

LL_CRC_GetOutputDataReverseMode

Function name

__STATIC_INLINE uint32_t LL_CRC_GetOutputDataReverseMode (const CRC_TypeDef * CRCx)

Function description

Return type of reversal of the bit order of the Output data.

Parameters

- **CRCx:** CRC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_CRC_OUTDATA_REVERSE_NONE
  – LL_CRC_OUTDATA_REVERSE_BIT

Reference Manual to LL API cross reference:

- CR REV_OUT LL_CRC_GetOutputDataReverseMode

LL_CRC_SetInitialData

Function name

__STATIC_INLINE void LL_CRC_SetInitialData (CRC_TypeDef * CRCx, uint32_t InitCrc)

Function description

Initialize the Programmable initial CRC value.

Parameters

- **CRCx:** CRC Instance
- **InitCrc:** Value to be programmed in Programmable initial CRC value register

Return values

- **None:**

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to write the correct value
- LL_CRC_DEFAULT_CRC_INITVALUE could be used as value for InitCrc parameter.

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_SetInitialData

LL_CRC_GetInitialData

Function name

__STATIC_INLINE uint32_t LL_CRC_GetInitialData (const CRC_TypeDef * CRCx)

Function description

Return current Initial CRC value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable initial CRC value register

Notes

- If the CRC size is less than 32 bits, the least significant bits are used to read the correct value

Reference Manual to LL API cross reference:

- INIT INIT LL_CRC_GetInitialData

LL_CRC_SetPolynomialCoef

Function name

__STATIC_INLINE void LL_CRC_SetPolynomialCoef (CRC_TypeDef * CRCx, uint32_t PolynomCoef)

Function description

Initialize the Programmable polynomial value (coefficients of the polynomial to be used for CRC calculation).

Parameters

- **CRCx:** CRC Instance
- **PolynomCoef:** Value to be programmed in Programmable Polynomial value register

Return values

- **None:**

Notes

- LL_CRC_DEFAULT_CRC32_POLY could be used as value for PolynomCoef parameter.
- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_SetPolynomialCoef

LL_CRC_GetPolynomialCoef

Function name

__STATIC_INLINE uint32_t LL_CRC_GetPolynomialCoef (const CRC_TypeDef * CRCx)

Function description

Return current Programmable polynomial value.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** programmed in Programmable Polynomial value register

Notes

- Please check Reference Manual and existing Errata Sheets, regarding possible limitations for Polynomial values usage. For example, for a polynomial of degree 7, X^7 + X^6 + X^5 + X^2 + 1 is written 0x65

Reference Manual to LL API cross reference:

- POL POL LL_CRC_GetPolynomialCoef

LL_CRC_FeedData32

Function name

__STATIC_INLINE void LL_CRC_FeedData32 (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Write given 32-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData32

LL_CRC_FeedData16

Function name

__STATIC_INLINE void LL_CRC_FeedData16 (CRC_TypeDef * CRCx, uint16_t InData)

Function description

Write given 16-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 16 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData16

LL_CRC_FeedData8

Function name

__STATIC_INLINE void LL_CRC_FeedData8 (CRC_TypeDef * CRCx, uint8_t InData)

Function description

Write given 8-bit data to the CRC calculator.

Parameters

- **CRCx:** CRC Instance
- **InData:** 8 bit value to be provided to CRC calculator between between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_CRC_FeedData8

LL_CRC_ReadData32

Function name

__STATIC_INLINE uint32_t LL_CRC_ReadData32 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (32 bits).

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData32

LL_CRC_ReadData16

Function name

__STATIC_INLINE uint16_t LL_CRC_ReadData16 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (16 bits).

Notes

- This function is expected to be used in a 16 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData16

LL_CRC_ReadData8

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData8 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (8 bits).

Notes

- This function is expected to be used in a 8 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData8

LL_CRC_ReadData7

Function name

__STATIC_INLINE uint8_t LL_CRC_ReadData7 (const CRC_TypeDef * CRCx)

Function description

Return current CRC calculation result.

Parameters

- **CRCx:** CRC Instance

Return values

- **Current:** CRC calculation result as stored in CRC_DR register (7 bits).

Notes

- This function is expected to be used in a 7 bits CRC polynomial size context.

Reference Manual to LL API cross reference:

- DR DR LL_CRC_ReadData7

LL_CRC_Read_IDR

Function name

__STATIC_INLINE uint32_t LL_CRC_Read_IDR (const CRC_TypeDef * CRCx)

Function description

Return data stored in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance

Return values

- **Value:** stored in CRC_IDR register (General-purpose 32-bit data register).

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Read_IDR

LL_CRC_Write_IDR

Function name

__STATIC_INLINE void LL_CRC_Write_IDR (CRC_TypeDef * CRCx, uint32_t InData)

Function description

Store data in the Independent Data(IDR) register.

Parameters

- **CRCx:** CRC Instance
- **InData:** value to be stored in CRC_IDR register (32-bit) between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- This register can be used as a temporary storage location for one 32-bit long data.

Reference Manual to LL API cross reference:

- IDR IDR LL_CRC_Write_IDR

LL_CRC_DeInit

Function name

ErrorStatus LL_CRC_DeInit (const CRC_TypeDef * CRCx)

Function description

De-initialize CRC registers (Registers restored to their default values).

Parameters

- **CRCx:** CRC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRC registers are de-initialized
  - ERROR: CRC registers are not de-initialized

95.2 CRC Firmware driver defines

The following section lists the various define and macros of the module.

95.2.1 CRC

CRC

Default CRC computation initialization value

LL_CRC_DEFAULT_CRC_INITVALUE

Default CRC computation initialization value

Default CRC generating polynomial value

LL_CRC_DEFAULT_CRC32_POLY

Default CRC generating polynomial value

Input Data Reverse

LL_CRC_INDATA_REVERSE_NONE

Input Data bit order not affected

LL_CRC_INDATA_REVERSE_BYTE

Input Data bit reversal done by byte

LL_CRC_INDATA_REVERSE_HALFWORD

Input Data bit reversal done by half-word

LL_CRC_INDATA_REVERSE_WORD

Input Data bit reversal done by word

Output Data Reverse

LL_CRC_OUTDATA_REVERSE_NONE

Output Data bit order not affected

LL_CRC_OUTDATA_REVERSE_BIT

Output Data bit reversal done by bit

Polynomial length

LL_CRC_POLYLENGTH_32B

32 bits Polynomial size

LL_CRC_POLYLENGTH_16B

16 bits Polynomial size

LL_CRC_POLYLENGTH_8B

8 bits Polynomial size

LL_CRC_POLYLENGTH_7B

7 bits Polynomial size

Common Write and read registers Macros

LL_CRC_WriteReg

Description:

- Write a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRC_ReadReg

Description:

- Read a value in CRC register.

Parameters:

- __INSTANCE__: CRC Instance
- __REG__: Register to be read

Return value:

- Register: value

96 LL CRS Generic Driver

96.1 CRS Firmware driver API description

The following section lists the various functions of the CRS library.

96.1.1 Detailed description of functions

LL_CRS_EnableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_EnableFreqErrorCounter (void)

Function description

Enable Frequency error counter.

Return values

- **None:**

Notes

- When this bit is set, the CRS_CFGR register is write-protected and cannot be modified

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_EnableFreqErrorCounter

LL_CRS_DisableFreqErrorCounter

Function name

__STATIC_INLINE void LL_CRS_DisableFreqErrorCounter (void)

Function description

Disable Frequency error counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_DisableFreqErrorCounter

LL_CRS_IsEnabledFreqErrorCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledFreqErrorCounter (void)

Function description

Check if Frequency error counter is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEN LL_CRS_IsEnabledFreqErrorCounter

LL_CRS_EnableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_EnableAutoTrimming (void)

Function description

Enable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_EnableAutoTrimming

LL_CRS_DisableAutoTrimming

Function name

__STATIC_INLINE void LL_CRS_DisableAutoTrimming (void)

Function description

Disable Automatic trimming counter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_DisableAutoTrimming

LL_CRS_IsEnabledAutoTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledAutoTrimming (void)

Function description

Check if Automatic trimming is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR AUTOTRIMEN LL_CRS_IsEnabledAutoTrimming

LL_CRS_SetHSI48SmoothTrimming

Function name

__STATIC_INLINE void LL_CRS_SetHSI48SmoothTrimming (uint32_t Value)

Function description

Set HSI48 oscillator smooth trimming.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 127

Return values

- **None:**

Notes

- When the AUTOTRIMEN bit is set, this field is controlled by hardware and is read-only
- Default value can be set thanks to LL_CRS_HSI48CALIBRATION_DEFAULT

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_SetHSI48SmoothTrimming

LL_CRS_GetHSI48SmoothTrimming

Function name

__STATIC_INLINE uint32_t LL_CRS_GetHSI48SmoothTrimming (void)

Function description

Get HSI48 oscillator smooth trimming.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 127

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_GetHSI48SmoothTrimming

LL_CRS_SetReloadCounter

Function name

__STATIC_INLINE void LL_CRS_SetReloadCounter (uint32_t Value)

Function description

Set counter reload value.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 0xFFFF

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_RELOADVALUE_DEFAULT Otherwise it can be calculated in using macro __LL_CRS_CALC_CALCULATE_RELOADVALUE (_FTARGET_, _FSYNC_)

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_SetReloadCounter

LL_CRS_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_CRS_GetReloadCounter (void)

Function description

Get counter reload value.

Return values

- **a:** number between Min_Data = 0 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- CFGR RELOAD LL_CRS_GetReloadCounter

LL_CRS_SetFreqErrorLimit

Function name

__STATIC_INLINE void LL_CRS_SetFreqErrorLimit (uint32_t Value)

Function description

Set frequency error limit.

Parameters

- **Value:** a number between Min_Data = 0 and Max_Data = 255

Return values

- **None:**

Notes

- Default value can be set thanks to LL_CRS_ERRORLIMIT_DEFAULT

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_SetFreqErrorLimit

LL_CRS_GetFreqErrorLimit

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorLimit (void)

Function description

Get frequency error limit.

Return values

- **A:** number between Min_Data = 0 and Max_Data = 255

Reference Manual to LL API cross reference:

- CFGR FELIM LL_CRS_GetFreqErrorLimit

LL_CRS_SetSyncDivider

Function name

__STATIC_INLINE void LL_CRS_SetSyncDivider (uint32_t Divider)

Function description

Set division factor for SYNC signal.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_SetSyncDivider

LL_CRS_GetSyncDivider

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncDivider (void)

Function description

Get division factor for SYNC signal.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_DIV_1
  - LL_CRS_SYNC_DIV_2
  - LL_CRS_SYNC_DIV_4
  - LL_CRS_SYNC_DIV_8
  - LL_CRS_SYNC_DIV_16
  - LL_CRS_SYNC_DIV_32
  - LL_CRS_SYNC_DIV_64
  - LL_CRS_SYNC_DIV_128

Reference Manual to LL API cross reference:

- CFGR SYNCDIV LL_CRS_GetSyncDivider

LL_CRS_SetSyncSignalSource

Function name

__STATIC_INLINE void LL_CRS_SetSyncSignalSource (uint32_t Source)

Function description

Set SYNC signal source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_SetSyncSignalSource

LL_CRS_GetSyncSignalSource

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncSignalSource (void)

Function description

Get SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_SOURCE_GPIO
  - LL_CRS_SYNC_SOURCE_LSE
  - LL_CRS_SYNC_SOURCE_USB

Reference Manual to LL API cross reference:

- CFGR SYNCSRC LL_CRS_GetSyncSignalSource

LL_CRS_SetSyncPolarity

Function name

__STATIC_INLINE void LL_CRS_SetSyncPolarity (uint32_t Polarity)

Function description

Set input polarity for the SYNC signal source.

Parameters

- **Polarity:** This parameter can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_SetSyncPolarity

LL_CRS_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_CRS_GetSyncPolarity (void)

Function description

Get input polarity for the SYNC signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_SYNC_POLARITY_RISING
  - LL_CRS_SYNC_POLARITY_FALLING

Reference Manual to LL API cross reference:

- CFGR SYNCPOL LL_CRS_GetSyncPolarity

LL_CRS_ConfigSynchronization

Function name

__STATIC_INLINE void LL_CRS_ConfigSynchronization (uint32_t HSI48CalibrationValue, uint32_t ErrorLimitValue, uint32_t ReloadValue, uint32_t Settings)

Function description

Configure CRS for the synchronization.

Parameters

- **HSI48CalibrationValue:** a number between Min_Data = 0 and Max_Data = 127
- **ErrorLimitValue:** a number between Min_Data = 0 and Max_Data = 0xFFFF
- **ReloadValue:** a number between Min_Data = 0 and Max_Data = 255
- **Settings:** This parameter can be a combination of the following values:
  - LL_CRS_SYNC_DIV_1 or LL_CRS_SYNC_DIV_2 or LL_CRS_SYNC_DIV_4 or LL_CRS_SYNC_DIV_8 or LL_CRS_SYNC_DIV_16 or LL_CRS_SYNC_DIV_32 or LL_CRS_SYNC_DIV_64 or LL_CRS_SYNC_DIV_128
  - LL_CRS_SYNC_SOURCE_GPIO or LL_CRS_SYNC_SOURCE_LSE or LL_CRS_SYNC_SOURCE_USB
  - LL_CRS_SYNC_POLARITY_RISING or LL_CRS_SYNC_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TRIM LL_CRS_ConfigSynchronization
- CFGR RELOAD LL_CRS_ConfigSynchronization
- CFGR FELIM LL_CRS_ConfigSynchronization
- CFGR SYNCDIV LL_CRS_ConfigSynchronization
- CFGR SYNCSRC LL_CRS_ConfigSynchronization
- CFGR SYNCPOL LL_CRS_ConfigSynchronization

LL_CRS_GenerateEvent_SWSYNC

Function name

__STATIC_INLINE void LL_CRS_GenerateEvent_SWSYNC (void)

Function description

Generate software SYNC event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SWSYNC LL_CRS_GenerateEvent_SWSYNC

LL_CRS_GetFreqErrorDirection

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorDirection (void)

Function description

Get the frequency error direction latched in the time of the last SYNC event.

Return values

- **Returned:** value can be one of the following values:
  - LL_CRS_FREQ_ERROR_DIR_UP
  - LL_CRS_FREQ_ERROR_DIR_DOWN

Reference Manual to LL API cross reference:

- ISR FEDIR LL_CRS_GetFreqErrorDirection

LL_CRS_GetFreqErrorCapture

Function name

__STATIC_INLINE uint32_t LL_CRS_GetFreqErrorCapture (void)

Function description

Get the frequency error counter value latched in the time of the last SYNC event.

Return values

- **A:** number between Min_Data = 0x0000 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- ISR FECAP LL_CRS_GetFreqErrorCapture

LL_CRS_IsActiveFlag_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCOK (void)

Function description

Check if SYNC event OK signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCOKF LL_CRS_IsActiveFlag_SYNCOK

LL_CRS_IsActiveFlag_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCWARN (void)

Function description

Check if SYNC warning signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCWARNF LL_CRS_IsActiveFlag_SYNCWARN

LL_CRS_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ERR (void)

Function description

Check if Synchronization or trimming error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ERRF LL_CRS_IsActiveFlag_ERR

LL_CRS_IsActiveFlag_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_ESYNC (void)

Function description

Check if Expected SYNC signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ESYNCF LL_CRS_IsActiveFlag_ESYNC

LL_CRS_IsActiveFlag_SYNCERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCERR (void)

Function description

Check if SYNC error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCERR LL_CRS_IsActiveFlag_SYNCERR

LL_CRS_IsActiveFlag_SYNCMISS

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_SYNCMISS (void)

Function description

Check if SYNC missed error signal occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SYNCMISS LL_CRS_IsActiveFlag_SYNCMISS

LL_CRS_IsActiveFlag_TRIMOVF

Function name

__STATIC_INLINE uint32_t LL_CRS_IsActiveFlag_TRIMOVF (void)

Function description

Check if Trimming overflow or underflow occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TRIMOVF LL_CRS_IsActiveFlag_TRIMOVF

LL_CRS_ClearFlag_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCOK (void)

Function description

Clear the SYNC event OK flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCOKC LL_CRS_ClearFlag_SYNCOK

LL_CRS_ClearFlag_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_SYNCWARN (void)

Function description

Clear the SYNC warning flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR SYNCWARNC LL_CRS_ClearFlag_SYNCWARN

LL_CRS_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ERR (void)

Function description

Clear TRIMOVF, SYNCMISS and SYNCERR bits and consequently also the ERR flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ERRC LL_CRS_ClearFlag_ERR

LL_CRS_ClearFlag_ESYNC

Function name

__STATIC_INLINE void LL_CRS_ClearFlag_ESYNC (void)

Function description

Clear Expected SYNC flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ESYNCC LL_CRS_ClearFlag_ESYNC

LL_CRS_EnableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCOK (void)

Function description

Enable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_EnableIT_SYNCOK

LL_CRS_DisableIT_SYNCOK

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCOK (void)

Function description

Disable SYNC event OK interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_DisableIT_SYNCOK

LL_CRS_IsEnabledIT_SYNCOK

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCOK (void)

Function description

Check if SYNC event OK interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCOKIE LL_CRS_IsEnabledIT_SYNCOK

LL_CRS_EnableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_EnableIT_SYNCWARN (void)

Function description

Enable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_EnableIT_SYNCWARN

LL_CRS_DisableIT_SYNCWARN

Function name

__STATIC_INLINE void LL_CRS_DisableIT_SYNCWARN (void)

Function description

Disable SYNC warning interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_DisableIT_SYNCWARN

LL_CRS_IsEnabledIT_SYNCWARN

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_SYNCWARN (void)

Function description

Check if SYNC warning interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SYNCWARNIE LL_CRS_IsEnabledIT_SYNCWARN

LL_CRS_EnableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ERR (void)

Function description

Enable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_EnableIT_ERR

LL_CRS_DisableIT_ERR

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ERR (void)

Function description

Disable Synchronization or trimming error interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_DisableIT_ERR

LL_CRS_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ERR (void)

Function description

Check if Synchronization or trimming error interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ERRIE LL_CRS_IsEnabledIT_ERR

LL_CRS_EnableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_EnableIT_ESYNC (void)

Function description

Enable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_EnableIT_ESYNC

LL_CRS_DisableIT_ESYNC

Function name

__STATIC_INLINE void LL_CRS_DisableIT_ESYNC (void)

Function description

Disable Expected SYNC interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_DisableIT_ESYNC

LL_CRS_IsEnabledIT_ESYNC

Function name

__STATIC_INLINE uint32_t LL_CRS_IsEnabledIT_ESYNC (void)

Function description

Check if Expected SYNC interrupt is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ESYNCIE LL_CRS_IsEnabledIT_ESYNC

LL_CRS_DeInit

Function name

ErrorStatus LL_CRS_DeInit (void)

Function description

De-Initializes CRS peripheral registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: CRS registers are de-initialized
  - ERROR: not applicable

96.2 CRS Firmware driver defines

The following section lists the various define and macros of the module.

96.2.1 CRS

CRS

Default Values

LL_CRS_RELOADVALUE_DEFAULT

Notes:

- The reset value of the RELOAD field corresponds to a target frequency of 48 MHz and a synchronization signal frequency of 1 kHz (SOF signal from USB)

LL_CRS_ERRORLIMIT_DEFAULT

LL_CRS_HSI48CALIBRATION_DEFAULT

Notes:

- The default value is 64, which corresponds to the middle of the trimming interval. The trimming step is specified in the product datasheet. A higher TRIM value corresponds to a higher output frequency.

Frequency Error Direction

LL_CRS_FREQ_ERROR_DIR_UP

Upcounting direction, the actual frequency is above the target

LL_CRS_FREQ_ERROR_DIR_DOWN

Downcounting direction, the actual frequency is below the target

Get Flags Defines

LL_CRS_ISR_SYNCOKF

LL_CRS_ISR_SYNCWARNF

LL_CRS_ISR_ERRF

LL_CRS_ISR_ESYNCF

LL_CRS_ISR_SYNCERR

LL_CRS_ISR_SYNCMISS

LL_CRS_ISR_TRIMOVF

IT Defines

LL_CRS_CR_SYNCOKIE

LL_CRS_CR_SYNCWARNIE

LL_CRS_CR_ERRIE

LL_CRS_CR_ESYNCIE

Synchronization Signal Divider

LL_CRS_SYNC_DIV_1

Synchro Signal not divided (default)

LL_CRS_SYNC_DIV_2

Synchro Signal divided by 2

LL_CRS_SYNC_DIV_4

Synchro Signal divided by 4

LL_CRS_SYNC_DIV_8

Synchro Signal divided by 8

LL_CRS_SYNC_DIV_16

Synchro Signal divided by 16

LL_CRS_SYNC_DIV_32

Synchro Signal divided by 32

LL_CRS_SYNC_DIV_64

Synchro Signal divided by 64

LL_CRS_SYNC_DIV_128

Synchro Signal divided by 128

Synchronization Signal Polarity

LL_CRS_SYNC_POLARITY_RISING

Synchro Active on rising edge (default)

LL_CRS_SYNC_POLARITY_FALLING

Synchro Active on falling edge

Synchronization Signal Source

LL_CRS_SYNC_SOURCE_GPIO

Synchro Signal source GPIO

LL_CRS_SYNC_SOURCE_LSE

Synchro Signal source LSE

LL_CRS_SYNC_SOURCE_USB

Synchro Signal source USB SOF (default)

Exported_Macros_Calculate_Reload

__LL_CRS_CALC_CALCULATE_RELOADVALUE

Description:

- Macro to calculate reload value to be set in CRS register according to target and sync frequencies.

Parameters:

- __FTARGET__: Target frequency (value in Hz)
- __FSYNC__: Synchronization signal frequency (value in Hz)

Return value:

- Reload: value (in Hz)

Notes:

- The RELOAD value should be selected according to the ratio between the target frequency and the frequency of the synchronization source after prescaling. It is then decreased by one in order to reach the expected synchronization on the zero value. The formula is the following: RELOAD = (fTARGET / fSYNC) -1

Common Write and read registers Macros

LL_CRS_WriteReg

Description:

- Write a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_CRS_ReadReg

Description:

- Read a value in CRS register.

Parameters:

- __INSTANCE__: CRS Instance
- __REG__: Register to be read

Return value:

- Register: value

97 LL DAC Generic Driver

97.1 DAC Firmware driver registers structures

97.1.1 LL_DAC_InitTypeDef

LL_DAC_InitTypeDef is defined in the stm32u5xx_ll_dac.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t WaveAutoGeneration***
- ***uint32_t WaveAutoGenerationConfig***
- ***uint32_t OutputBuffer***
- ***uint32_t OutputConnection***
- ***uint32_t OutputMode***

Field Documentation

- ***uint32_t LL_DAC_InitTypeDef::TriggerSource***
  Set the conversion trigger source for the selected DAC channel: internal (SW start) or from external peripheral (timer event, external interrupt line). This parameter can be a value of ***DAC_LL_EC_TRIGGER_SOURCE***This feature can be modified afterwards using unitary function **LL_DAC_SetTriggerSource()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGeneration***
  Set the waveform automatic generation mode for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_WAVE_AUTO_GENERATION_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetWaveAutoGeneration()**.
- ***uint32_t LL_DAC_InitTypeDef::WaveAutoGenerationConfig***
  Set the waveform automatic generation mode for the selected DAC channel. If waveform automatic generation mode is set to noise, this parameter can be a value of ***DAC_LL_EC_WAVE_NOISE_LFSR_UNMASK_BITS*** If waveform automatic generation mode is set to triangle, this parameter can be a value of ***DAC_LL_EC_WAVE_TRIANGLE_AMPLITUDE***
  Note:
  – If waveform automatic generation mode is disabled, this parameter is discarded.

  This feature can be modified afterwards using unitary function **LL_DAC_SetWaveNoiseLFSR()**, **LL_DAC_SetWaveTriangleAmplitude()** depending on the wave automatic generation selected.
- ***uint32_t LL_DAC_InitTypeDef::OutputBuffer***
  Set the output buffer for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_BUFFER***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputBuffer()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputConnection***
  Set the output connection for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_CONNECTION***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputConnection()**.
- ***uint32_t LL_DAC_InitTypeDef::OutputMode***
  Set the output mode normal or sample-and-hold for the selected DAC channel. This parameter can be a value of ***DAC_LL_EC_OUTPUT_MODE***This feature can be modified afterwards using unitary function **LL_DAC_SetOutputMode()**.

97.2 DAC Firmware driver API description

The following section lists the various functions of the DAC library.

97.2.1 Detailed description of functions

LL_DAC_SetHighFrequencyMode

Function name

__STATIC_INLINE void LL_DAC_SetHighFrequencyMode (DAC_TypeDef * DACx, uint32_t HighFreqMode)

Function description

Set the high frequency interface mode for the selected DAC instance.

Parameters

- **DACx:** DAC instance
- **HighFreqMode:** This parameter can be one of the following values:
  – LL_DAC_HIGH_FREQ_MODE_DISABLE
  – LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
  – LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR HFSEL LL_DAC_SetHighFrequencyMode

LL_DAC_GetHighFrequencyMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetHighFrequencyMode (const DAC_TypeDef * DACx)

Function description

Get the high frequency interface mode for the selected DAC instance.

Parameters

- **DACx:** DAC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DAC_HIGH_FREQ_MODE_DISABLE
  – LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
  – LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ

Reference Manual to LL API cross reference:

- MCR HFSEL LL_DAC_GetHighFrequencyMode

LL_DAC_SetMode

Function name

__STATIC_INLINE void LL_DAC_SetMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t ChannelMode)

Function description

Set the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **ChannelMode:** This parameter can be one of the following values:
  – LL_DAC_MODE_NORMAL_OPERATION
  – LL_DAC_MODE_CALIBRATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_SetMode
- CR CEN2 LL_DAC_SetMode

LL_DAC_GetMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetMode (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the operating mode for the selected DAC channel: calibration or normal operating mode.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  – LL_DAC_MODE_NORMAL_OPERATION
  – LL_DAC_MODE_CALIBRATION

Reference Manual to LL API cross reference:

- CR CEN1 LL_DAC_GetMode
- CR CEN2 LL_DAC_GetMode

LL_DAC_SetTrimmingValue

Function name

__STATIC_INLINE void LL_DAC_SetTrimmingValue (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TrimmingValue)

Function description

Set the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_SetTrimmingValue
- CCR OTRIM2 LL_DAC_SetTrimmingValue

LL_DAC_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTrimmingValue (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the offset trimming value for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **TrimmingValue:** Value between Min_Data=0x00 and Max_Data=0x1F

Reference Manual to LL API cross reference:

- CCR OTRIM1 LL_DAC_GetTrimmingValue
- CCR OTRIM2 LL_DAC_GetTrimmingValue

LL_DAC_SetTriggerSource

Function name

__STATIC_INLINE void LL_DAC_SetTriggerSource (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriggerSource)

Function description

Set the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **TriggerSource:** This parameter can be one of the following values:
  – LL_DAC_TRIG_SOFTWARE
  – LL_DAC_TRIG_EXT_TIM1_TRGO
  – LL_DAC_TRIG_EXT_TIM2_TRGO
  – LL_DAC_TRIG_EXT_TIM4_TRGO
  – LL_DAC_TRIG_EXT_TIM5_TRGO
  – LL_DAC_TRIG_EXT_TIM6_TRGO
  – LL_DAC_TRIG_EXT_TIM7_TRGO
  – LL_DAC_TRIG_EXT_TIM8_TRGO
  – LL_DAC_TRIG_EXT_TIM15_TRGO
  – LL_DAC_TRIG_EXT_LPTIM1_CH1
  – LL_DAC_TRIG_EXT_LPTIM3_CH1
  – LL_DAC_TRIG_EXT_EXTI_LINE9

Return values

- **None:**

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- To set conversion trigger source, DAC channel must be disabled. Otherwise, the setting is discarded.
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_SetTriggerSource
- CR TSEL2 LL_DAC_SetTriggerSource

LL_DAC_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_DAC_GetTriggerSource (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the conversion trigger source for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIG_SOFTWARE
  - LL_DAC_TRIG_EXT_TIM1_TRGO
  - LL_DAC_TRIG_EXT_TIM2_TRGO
  - LL_DAC_TRIG_EXT_TIM4_TRGO
  - LL_DAC_TRIG_EXT_TIM5_TRGO
  - LL_DAC_TRIG_EXT_TIM6_TRGO
  - LL_DAC_TRIG_EXT_TIM7_TRGO
  - LL_DAC_TRIG_EXT_TIM8_TRGO
  - LL_DAC_TRIG_EXT_TIM15_TRGO
  - LL_DAC_TRIG_EXT_LPTIM1_CH1
  - LL_DAC_TRIG_EXT_LPTIM3_CH1
  - LL_DAC_TRIG_EXT_EXTI_LINE9

Notes

- For conversion trigger source to be effective, DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- Availability of parameters of trigger sources from timer depends on timers availability on the selected device.

Reference Manual to LL API cross reference:

- CR TSEL1 LL_DAC_GetTriggerSource
- CR TSEL2 LL_DAC_GetTriggerSource

LL_DAC_SetWaveAutoGeneration

Function name

__STATIC_INLINE void LL_DAC_SetWaveAutoGeneration (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t WaveAutoGeneration)

Function description

Set the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **WaveAutoGeneration:** This parameter can be one of the following values:
  - LL_DAC_WAVE_AUTO_GENERATION_NONE
  - LL_DAC_WAVE_AUTO_GENERATION_NOISE
  - LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_SetWaveAutoGeneration
- CR WAVE2 LL_DAC_SetWaveAutoGeneration

LL_DAC_GetWaveAutoGeneration

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveAutoGeneration (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the waveform automatic generation mode for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  – LL_DAC_WAVE_AUTO_GENERATION_NONE
  – LL_DAC_WAVE_AUTO_GENERATION_NOISE
  – LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

Reference Manual to LL API cross reference:

- CR WAVE1 LL_DAC_GetWaveAutoGeneration
- CR WAVE2 LL_DAC_GetWaveAutoGeneration

LL_DAC_SetWaveNoiseLFSR

Function name

__STATIC_INLINE void LL_DAC_SetWaveNoiseLFSR (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t NoiseLFSRMask)

Function description

Set the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **NoiseLFSRMask:** This parameter can be one of the following values:
  – LL_DAC_NOISE_LFSR_UNMASK_BIT0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  – LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveNoiseLFSR
- CR MAMP2 LL_DAC_SetWaveNoiseLFSR

LL_DAC_GetWaveNoiseLFSR

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveNoiseLFSR (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the noise waveform generation for the selected DAC channel: Noise mode and parameters LFSR (linear feedback shift register).

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_NOISE_LFSR_UNMASK_BIT0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS1_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS2_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS3_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS4_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS5_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS6_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS7_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS8_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS9_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS10_0
  - LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveNoiseLFSR
- CR MAMP2 LL_DAC_GetWaveNoiseLFSR

LL_DAC_SetWaveTriangleAmplitude

Function name

__STATIC_INLINE void LL_DAC_SetWaveTriangleAmplitude (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t TriangleAmplitude)

Function description

Set the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **TriangleAmplitude:** This parameter can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Return values

- **None:**

Notes

- For wave generation to be effective, DAC channel wave generation mode must be enabled using function LL_DAC_SetWaveAutoGeneration().
- This setting can be set when the selected DAC channel is disabled (otherwise, the setting operation is ignored).

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_SetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_SetWaveTriangleAmplitude

LL_DAC_GetWaveTriangleAmplitude

Function name

__STATIC_INLINE uint32_t LL_DAC_GetWaveTriangleAmplitude (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the triangle waveform generation for the selected DAC channel: triangle mode and amplitude.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_TRIANGLE_AMPLITUDE_1
  - LL_DAC_TRIANGLE_AMPLITUDE_3
  - LL_DAC_TRIANGLE_AMPLITUDE_7
  - LL_DAC_TRIANGLE_AMPLITUDE_15
  - LL_DAC_TRIANGLE_AMPLITUDE_31
  - LL_DAC_TRIANGLE_AMPLITUDE_63
  - LL_DAC_TRIANGLE_AMPLITUDE_127
  - LL_DAC_TRIANGLE_AMPLITUDE_255
  - LL_DAC_TRIANGLE_AMPLITUDE_511
  - LL_DAC_TRIANGLE_AMPLITUDE_1023
  - LL_DAC_TRIANGLE_AMPLITUDE_2047
  - LL_DAC_TRIANGLE_AMPLITUDE_4095

Reference Manual to LL API cross reference:

- CR MAMP1 LL_DAC_GetWaveTriangleAmplitude
- CR MAMP2 LL_DAC_GetWaveTriangleAmplitude

LL_DAC_ConfigOutput

Function name

__STATIC_INLINE void LL_DAC_ConfigOutput (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode, uint32_t OutputBuffer, uint32_t OutputConnection)

Function description

Set the output for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **OutputMode:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_MODE_NORMAL
  - LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD
- **OutputBuffer:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_BUFFER_ENABLE
  - LL_DAC_OUTPUT_BUFFER_DISABLE
- **OutputConnection:** This parameter can be one of the following values:
  - LL_DAC_OUTPUT_CONNECT_GPIO
  - LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- This function set several features: mode normal or sample-and-holdbufferconnection to GPIO or internal path. These features can also be set individually using dedicated functions:LL_DAC_SetOutputBuffer()LL_DAC_SetOutputMode()LL_DAC_SetOutputConnection()
- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).
- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_ConfigOutput
- CR MODE2 LL_DAC_ConfigOutput

LL_DAC_SetOutputMode

Function name

__STATIC_INLINE void LL_DAC_SetOutputMode (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputMode)

Function description

Set the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **OutputMode:** This parameter can be one of the following values:
  – LL_DAC_OUTPUT_MODE_NORMAL
  – LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Return values

- **None:**

Notes

- Mode sample-and-hold requires an external capacitor to be connected between DAC channel output and ground. Capacitor value depends on load on DAC channel output and sample-and-hold timings configured. As indication, capacitor typical value is 100nF (refer to device datasheet, parameter "CSH").

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputMode
- CR MODE2 LL_DAC_SetOutputMode

LL_DAC_GetOutputMode

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputMode (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output mode normal or sample-and-hold for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  – LL_DAC_OUTPUT_MODE_NORMAL
  – LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputMode
- CR MODE2 LL_DAC_GetOutputMode

LL_DAC_SetOutputBuffer

Function name

__STATIC_INLINE void LL_DAC_SetOutputBuffer (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputBuffer)

Function description

Set the output buffer for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **OutputBuffer:** This parameter can be one of the following values:
  – LL_DAC_OUTPUT_BUFFER_ENABLE
  – LL_DAC_OUTPUT_BUFFER_DISABLE

Return values

- **None:**

Notes

- On this STM32 series, when buffer is enabled, its offset can be trimmed: factory calibration default values can be replaced by user trimming values, using function LL_DAC_SetTrimmingValue().

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputBuffer
- CR MODE2 LL_DAC_SetOutputBuffer

LL_DAC_GetOutputBuffer

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputBuffer (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output buffer state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  – LL_DAC_OUTPUT_BUFFER_ENABLE
  – LL_DAC_OUTPUT_BUFFER_DISABLE

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputBuffer
- CR MODE2 LL_DAC_GetOutputBuffer

LL_DAC_SetOutputConnection

Function name

__STATIC_INLINE void LL_DAC_SetOutputConnection (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t OutputConnection)

Function description

Set the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **OutputConnection:** This parameter can be one of the following values:
  – LL_DAC_OUTPUT_CONNECT_GPIO
  – LL_DAC_OUTPUT_CONNECT_INTERNAL

Return values

- **None:**

Notes

- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_SetOutputConnection
- CR MODE2 LL_DAC_SetOutputConnection

LL_DAC_GetOutputConnection

Function name

__STATIC_INLINE uint32_t LL_DAC_GetOutputConnection (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the output connection for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  – LL_DAC_OUTPUT_CONNECT_GPIO
  – LL_DAC_OUTPUT_CONNECT_INTERNAL

Notes

- On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. if output connection is set to internal path and output buffer is enabled (whatever output mode): output connection is also connected to GPIO pin (both connections to GPIO pin and internal path).if output connection is set to GPIO pin, output buffer is disabled, output mode set to sample and hold: output connection is also connected to internal path (both connections to GPIO pin and internal path).

Reference Manual to LL API cross reference:

- CR MODE1 LL_DAC_GetOutputConnection
- CR MODE2 LL_DAC_GetOutputConnection

LL_DAC_SetSampleAndHoldSampleTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldSampleTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SampleTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **SampleTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Notes

- Sample time must be set when DAC channel is disabled or during DAC operation when DAC channel flag BWSTx is reset, otherwise the setting is ignored. Check BWSTx flag state using function "LL_DAC_IsActiveFlag_BWSTx()".

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_SetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_SetSampleAndHoldSampleTime

LL_DAC_GetSampleAndHoldSampleTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldSampleTime (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: sample time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHSR1 TSAMPLE1 LL_DAC_GetSampleAndHoldSampleTime
- SHSR2 TSAMPLE2 LL_DAC_GetSampleAndHoldSampleTime

LL_DAC_SetSampleAndHoldHoldTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldHoldTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t HoldTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **HoldTime:** Value between Min_Data=0x000 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_SetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_SetSampleAndHoldHoldTime

LL_DAC_GetSampleAndHoldHoldTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldHoldTime (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: hold time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- SHHR THOLD1 LL_DAC_GetSampleAndHoldHoldTime
- SHHR THOLD2 LL_DAC_GetSampleAndHoldHoldTime

LL_DAC_SetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE void LL_DAC_SetSampleAndHoldRefreshTime (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t RefreshTime)

Function description

Set the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **RefreshTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_SetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_SetSampleAndHoldRefreshTime

LL_DAC_GetSampleAndHoldRefreshTime

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSampleAndHoldRefreshTime (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the sample-and-hold timing for the selected DAC channel: refresh time.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- SHRR TREFRESH1 LL_DAC_GetSampleAndHoldRefreshTime
- SHRR TREFRESH2 LL_DAC_GetSampleAndHoldRefreshTime

LL_DAC_SetSignedFormat

Function name

__STATIC_INLINE void LL_DAC_SetSignedFormat (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t SignedFormat)

Function description

Set the signed format for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **SignedFormat:** This parameter can be one of the following values:
  - LL_DAC_SIGNED_FORMAT_ENABLE
  - LL_DAC_SIGNED_FORMAT_DISABLE

Return values

- **None:**

Notes

- On this STM32 series, signed format can be used to inject Q1.15, Q1.11, Q1.7 signed format data to DAC. Ex when using 12bits data format (Q1.11 is used): 0x800 will output 0v level 0xFFF will output mid-scale level 0x000 will output mid-scale level 0x7FF will output full-scale level

Reference Manual to LL API cross reference:

- MCR SINFORMAT1 LL_DAC_SetSignedFormat
- MCR SINFORMAT2 LL_DAC_SetSignedFormat

LL_DAC_GetSignedFormat

Function name

__STATIC_INLINE uint32_t LL_DAC_GetSignedFormat (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get the signed format state for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **Returned:** value can be one of the following values:
  - LL_DAC_SIGNED_FORMAT_ENABLE
  - LL_DAC_SIGNED_FORMAT_DISABLE

Reference Manual to LL API cross reference:

- MCR SINFORMAT1 LL_DAC_GetSignedFormat
- MCR SINFORMAT2 LL_DAC_GetSignedFormat

LL_DAC_EnableDMAReq

Function name

__STATIC_INLINE void LL_DAC_EnableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_EnableDMAReq
- CR DMAEN2 LL_DAC_EnableDMAReq

LL_DAC_DisableDMAReq

Function name

__STATIC_INLINE void LL_DAC_DisableDMAReq (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA transfer request of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- To configure DMA source address (peripheral address), use function LL_DAC_DMA_GetRegAddr().

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_DisableDMAReq
- CR DMAEN2 LL_DAC_DisableDMAReq

LL_DAC_IsDMAReqEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMAReqEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA transfer request state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAEN1 LL_DAC_IsDMAReqEnabled
- CR DMAEN2 LL_DAC_IsDMAReqEnabled

LL_DAC_EnableDMADoubleDataMode

Function name

__STATIC_INLINE void LL_DAC_EnableDMADoubleDataMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC DMA Double data mode of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_EnableDMADoubleDataMode
- MCR DMADOUBLE2 LL_DAC_EnableDMADoubleDataMode

LL_DAC_DisableDMADoubleDataMode

Function name

__STATIC_INLINE void LL_DAC_DisableDMADoubleDataMode (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC DMA Double data mode of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_DisableDMADoubleDataMode
- MCR DMADOUBLE2 LL_DAC_DisableDMADoubleDataMode

LL_DAC_IsDMADoubleDataModeEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsDMADoubleDataModeEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC DMA double data mode state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- MCR DMADOUBLE1 LL_DAC_IsDMADoubleDataModeEnabled
- MCR DMADOUBLE2 LL_DAC_IsDMADoubleDataModeEnabled

LL_DAC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_DAC_DMA_GetRegAddr (const DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Register)

Function description

Function to help to configure DMA transfer to DAC: retrieve the DAC register address from DAC instance and a list of DAC registers intended to be used (most commonly) with DMA transfer.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **Register:** This parameter can be one of the following values:
  – LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED
  – LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED
  – LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

Return values

- **DAC:** register address

Notes

- These DAC registers are data holding registers: when DAC conversion is requested, DAC generates a DMA transfer request to have data available in DAC data holding registers.
- This macro is intended to be used with LL DMA driver, refer to function "LL_DMA_ConfigAddresses()". Example: LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1, (uint32_t)&< array or variable >, LL_DAC_DMA_GetRegAddr(DAC1, LL_DAC_CHANNEL_1, LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED), LL_DMA_DIRECTION_MEMORY_TO_PERIPH);

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12L1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR8R1 DACC1DHR LL_DAC_DMA_GetRegAddr
- DHR12R2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR12L2 DACC2DHR LL_DAC_DMA_GetRegAddr
- DHR8R2 DACC2DHR LL_DAC_DMA_GetRegAddr

LL_DAC_Enable

Function name

__STATIC_INLINE void LL_DAC_Enable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- After enable from off state, DAC channel requires a delay for output voltage to reach accuracy +/- 1 LSB. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Enable
- CR EN2 LL_DAC_Enable

LL_DAC_Disable

Function name

__STATIC_INLINE void LL_DAC_Disable (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_Disable
- CR EN2 LL_DAC_Disable

LL_DAC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC enable state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN1 LL_DAC_IsEnabled
- CR EN2 LL_DAC_IsEnabled

LL_DAC_IsReady

Function name

__STATIC_INLINE uint32_t LL_DAC_IsReady (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC ready for conversion state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC1RDY LL_DAC_IsReady
- SR DAC2RDY LL_DAC_IsReady

LL_DAC_EnableTrigger

Function name

__STATIC_INLINE void LL_DAC_EnableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Enable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- - If DAC trigger is disabled, DAC conversion is performed automatically once the data holding register is updated, using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ... If DAC trigger is enabled, DAC conversion is performed only when a hardware of software trigger event is occurring. Select trigger source using function LL_DAC_SetTriggerSource().

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_EnableTrigger
- CR TEN2 LL_DAC_EnableTrigger

LL_DAC_DisableTrigger

Function name

__STATIC_INLINE void LL_DAC_DisableTrigger (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Disable DAC trigger of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_DisableTrigger
- CR TEN2 LL_DAC_DisableTrigger

LL_DAC_IsTriggerEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsTriggerEnabled (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Get DAC trigger state of the selected channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEN1 LL_DAC_IsTriggerEnabled
- CR TEN2 LL_DAC_IsTriggerEnabled

LL_DAC_TrigSWConversion

Function name

__STATIC_INLINE void LL_DAC_TrigSWConversion (DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Trig DAC conversion by software for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can a combination of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **None:**

Notes

- Preliminarily, DAC trigger must be set to software trigger using function LL_DAC_Init() LL_DAC_SetTriggerSource() with parameter "LL_DAC_TRIGGER_SOFTWARE". and DAC trigger must be enabled using function LL_DAC_EnableTrigger().
- For devices featuring DAC with 2 channels: this function can perform a SW start of both DAC channels simultaneously. Two channels can be selected as parameter. Example: (LL_DAC_CHANNEL_1 | LL_DAC_CHANNEL_2)

Reference Manual to LL API cross reference:

- SWTRIGR SWTRIG1 LL_DAC_TrigSWConversion
- SWTRIGR SWTRIG2 LL_DAC_TrigSWConversion

LL_DAC_ConvertData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12R1 DACC1DHR LL_DAC_ConvertData12RightAligned
- DHR12R2 DACC2DHR LL_DAC_ConvertData12RightAligned

LL_DAC_ConvertData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData12LeftAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12L1 DACC1DHR LL_DAC_ConvertData12LeftAligned
- DHR12L2 DACC2DHR LL_DAC_ConvertData12LeftAligned

LL_DAC_ConvertData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertData8RightAligned (DAC_TypeDef * DACx, uint32_t DAC_Channel, uint32_t Data)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  - LL_DAC_CHANNEL_1
  - LL_DAC_CHANNEL_2
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8R1 DACC1DHR LL_DAC_ConvertData8RightAligned
- DHR8R2 DACC2DHR LL_DAC_ConvertData8RightAligned

LL_DAC_ConvertDualData12RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12RD DACC1DHR LL_DAC_ConvertDualData12RightAligned
- DHR12RD DACC2DHR LL_DAC_ConvertDualData12RightAligned

LL_DAC_ConvertDualData12LeftAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData12LeftAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 12 bits left alignment (MSB aligned on bit 15), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x000 and Max_Data=0xFFF
- **DataChannel2:** Value between Min_Data=0x000 and Max_Data=0xFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR12LD DACC1DHR LL_DAC_ConvertDualData12LeftAligned
- DHR12LD DACC2DHR LL_DAC_ConvertDualData12LeftAligned

LL_DAC_ConvertDualData8RightAligned

Function name

__STATIC_INLINE void LL_DAC_ConvertDualData8RightAligned (DAC_TypeDef * DACx, uint32_t DataChannel1, uint32_t DataChannel2)

Function description

Set the data to be loaded in the data holding register in format 8 bits left alignment (LSB aligned on bit 0), for both DAC channels.

Parameters

- **DACx:** DAC instance
- **DataChannel1:** Value between Min_Data=0x00 and Max_Data=0xFF
- **DataChannel2:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DHR8RD DACC1DHR LL_DAC_ConvertDualData8RightAligned
- DHR8RD DACC2DHR LL_DAC_ConvertDualData8RightAligned

LL_DAC_RetrieveOutputData

Function name

__STATIC_INLINE uint32_t LL_DAC_RetrieveOutputData (const DAC_TypeDef * DACx, uint32_t DAC_Channel)

Function description

Retrieve output data currently generated for the selected DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return values

- **Value:** between Min_Data=0x000 and Max_Data=0xFFF

Notes

- Whatever alignment and resolution settings (using functions "LL_DAC_ConvertData{8; 12}{Right; Left} Aligned()": LL_DAC_ConvertData12RightAligned(), ...), output data format is 12 bits right aligned (LSB aligned on bit 0).

Reference Manual to LL API cross reference:

- DOR1 DACC1DOR LL_DAC_RetrieveOutputData
- DOR2 DACC2DOR LL_DAC_RetrieveOutputData

LL_DAC_IsActiveFlag_CAL1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL1 (const DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG1 LL_DAC_IsActiveFlag_CAL1

LL_DAC_IsActiveFlag_CAL2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_CAL2 (const DAC_TypeDef * DACx)

Function description

Get DAC calibration offset flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CAL_FLAG2 LL_DAC_IsActiveFlag_CAL2

LL_DAC_IsActiveFlag_BWST1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST1 (const DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST1 LL_DAC_IsActiveFlag_BWST1

LL_DAC_IsActiveFlag_BWST2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_BWST2 (const DAC_TypeDef * DACx)

Function description

Get DAC busy writing sample time flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BWST2 LL_DAC_IsActiveFlag_BWST2

LL_DAC_IsActiveFlag_DAC1RDY

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DAC1RDY (const DAC_TypeDef * DACx)

Function description

Get DAC ready status flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC1RDY LL_DAC_IsActiveFlag_DAC1RDY

LL_DAC_IsActiveFlag_DAC2RDY

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DAC2RDY (const DAC_TypeDef * DACx)

Function description

Get DAC ready status flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DAC2RDY LL_DAC_IsActiveFlag_DAC2RDY

LL_DAC_IsActiveFlag_DORSTAT1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DORSTAT1 (const DAC_TypeDef * DACx)

Function description

Get DAC output register status flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DORSTAT1 LL_DAC_IsActiveFlag_DORSTAT1

LL_DAC_IsActiveFlag_DORSTAT2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DORSTAT2 (const DAC_TypeDef * DACx)

Function description

Get DAC output register status flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DORSTAT2 LL_DAC_IsActiveFlag_DORSTAT2

LL_DAC_IsActiveFlag_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR1 (const DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_IsActiveFlag_DMAUDR1

LL_DAC_IsActiveFlag_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsActiveFlag_DMAUDR2 (const DAC_TypeDef * DACx)

Function description

Get DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_IsActiveFlag_DMAUDR2

LL_DAC_ClearFlag_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR1 LL_DAC_ClearFlag_DMAUDR1

LL_DAC_ClearFlag_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_ClearFlag_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Clear DAC underrun flag for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR DMAUDR2 LL_DAC_ClearFlag_DMAUDR2

LL_DAC_EnableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_EnableIT_DMAUDR1

LL_DAC_EnableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_EnableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Enable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_EnableIT_DMAUDR2

LL_DAC_DisableIT_DMAUDR1

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR1 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_DisableIT_DMAUDR1

LL_DAC_DisableIT_DMAUDR2

Function name

__STATIC_INLINE void LL_DAC_DisableIT_DMAUDR2 (DAC_TypeDef * DACx)

Function description

Disable DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_DisableIT_DMAUDR2

LL_DAC_IsEnabledIT_DMAUDR1

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR1 (const DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 1.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE1 LL_DAC_IsEnabledIT_DMAUDR1

LL_DAC_IsEnabledIT_DMAUDR2

Function name

__STATIC_INLINE uint32_t LL_DAC_IsEnabledIT_DMAUDR2 (const DAC_TypeDef * DACx)

Function description

Get DMA underrun interrupt for DAC channel 2.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAUDRIE2 LL_DAC_IsEnabledIT_DMAUDR2

LL_DAC_EnableAutonomousMode

Function name

__STATIC_INLINE void LL_DAC_EnableAutonomousMode (DAC_TypeDef * DACx)

Function description

Enable DAC autonomous mode.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR AUTOMODE LL_DAC_EnableAutonomousMode

LL_DAC_DisableAutonomousMode

Function name

__STATIC_INLINE void LL_DAC_DisableAutonomousMode (DAC_TypeDef * DACx)

Function description

Disable DAC autonomous mode.

Parameters

- **DACx:** DAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR AUTOMODE LL_DAC_DisableAutonomousMode

LL_DAC_IsAutonomousModeEnabled

Function name

__STATIC_INLINE uint32_t LL_DAC_IsAutonomousModeEnabled (const DAC_TypeDef * DACx)

Function description

Get DAC autonomous mode state.

Parameters

- **DACx:** DAC instance

Return values

- **State:** of bit (1 or 0).

LL_DAC_DeInit

Function name

ErrorStatus LL_DAC_DeInit (const DAC_TypeDef * DACx)

Function description

De-initialize registers of the selected DAC instance to their default reset values.

Parameters

- **DACx:** DAC instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: DAC registers are de-initialized
  – ERROR: not applicable

LL_DAC_Init

Function name

ErrorStatus LL_DAC_Init (DAC_TypeDef * DACx, uint32_t DAC_Channel, const LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Initialize some features of DAC channel.

Parameters

- **DACx:** DAC instance
- **DAC_Channel:** This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2
- **DAC_InitStruct:** Pointer to a LL_DAC_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: DAC registers are initialized
  – ERROR: DAC registers are not initialized

Notes

- LL_DAC_Init() aims to ease basic configuration of a DAC channel. Leaving it ready to be enabled and output: a level by calling one of LL_DAC_ConvertData12RightAligned LL_DAC_ConvertData12LeftAligned LL_DAC_ConvertData8RightAligned or one of the supported autogenerated wave.
- This function allows configuration of: Output modeTriggerWave generation
- The setting of these parameters by function LL_DAC_Init() is conditioned to DAC state: DAC channel must be disabled.

LL_DAC_StructInit

Function name

void LL_DAC_StructInit (LL_DAC_InitTypeDef * DAC_InitStruct)

Function description

Set each LL_DAC_InitTypeDef field to default value.

Parameters

- **DAC_InitStruct:** pointer to a LL_DAC_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

97.3 DAC Firmware driver defines

The following section lists the various define and macros of the module.

97.3.1 DAC

DAC

DAC channels

LL_DAC_CHANNEL_1
DAC channel 1

LL_DAC_CHANNEL_2
DAC channel 2

DAC flags

LL_DAC_FLAG_DMAUDR1
DAC channel 1 flag DMA underrun

LL_DAC_FLAG_CAL1
DAC channel 1 flag offset calibration status

LL_DAC_FLAG_BWST1
DAC channel 1 flag busy writing sample time

LL_DAC_FLAG_DAC1RDY
DAC channel 1 flag ready

LL_DAC_FLAG_DORSTAT1
DAC channel 1 flag output register

LL_DAC_FLAG_DMAUDR2
DAC channel 2 flag DMA underrun

LL_DAC_FLAG_CAL2
DAC channel 2 flag offset calibration status

LL_DAC_FLAG_BWST2
DAC channel 2 flag busy writing sample time

LL_DAC_FLAG_DAC2RDY
DAC channel 2 flag ready

LL_DAC_FLAG_DORSTAT2
DAC channel 2 flag output register

DAC high frequency interface mode

LL_DAC_HIGH_FREQ_MODE_DISABLE
High frequency interface mode disabled

LL_DAC_HIGH_FREQ_MODE_ABOVE_80MHZ
High frequency interface mode compatible to AHB>80MHz enabled

LL_DAC_HIGH_FREQ_MODE_ABOVE_160MHZ
High frequency interface mode compatible to AHB>160MHz enabled

Definitions of DAC hardware constraints delays

LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US
Delay for DAC channel voltage settling time from DAC channel startup (transition from disable to enable)

LL_DAC_DELAY_VOLTAGE_SETTLING_US
Delay for DAC channel voltage settling time

DAC interruptions

LL_DAC_IT_DMAUDRIE1

DAC channel 1 interruption DMA underrun

LL_DAC_IT_DMAUDRIE2

DAC channel 2 interruption DMA underrun

DAC operating mode

LL_DAC_MODE_NORMAL_OPERATION

DAC channel in mode normal operation

LL_DAC_MODE_CALIBRATION

DAC channel in mode calibration

DAC channel output buffer

LL_DAC_OUTPUT_BUFFER_ENABLE

The selected DAC channel output is buffered: higher drive current capability, but also higher current consumption

LL_DAC_OUTPUT_BUFFER_DISABLE

The selected DAC channel output is not buffered: lower drive current capability, but also lower current consumption

DAC channel output connection

LL_DAC_OUTPUT_CONNECT_GPIO

The selected DAC channel output is connected to external pin

LL_DAC_OUTPUT_CONNECT_INTERNAL

The selected DAC channel output is connected to on-chip peripherals via internal paths. On this STM32 series, output connection depends on output mode (normal or sample and hold) and output buffer state. Refer to comments of function

DAC channel output mode

LL_DAC_OUTPUT_MODE_NORMAL

The selected DAC channel output is on mode normal.

LL_DAC_OUTPUT_MODE_SAMPLE_AND_HOLD

The selected DAC channel output is on mode sample-and-hold. Mode sample-and-hold requires an external capacitor, refer to description of function

DAC registers compliant with specific purpose

LL_DAC_DMA_REG_DATA_12BITS_RIGHT_ALIGNED

DAC channel data holding register 12 bits right aligned

LL_DAC_DMA_REG_DATA_12BITS_LEFT_ALIGNED

DAC channel data holding register 12 bits left aligned

LL_DAC_DMA_REG_DATA_8BITS_RIGHT_ALIGNED

DAC channel data holding register 8 bits right aligned

DAC channel output resolution

LL_DAC_RESOLUTION_12B

DAC channel resolution 12 bits

LL_DAC_RESOLUTION_8B

DAC channel resolution 8 bits

DAC channel signed format

LL_DAC_SIGNED_FORMAT_DISABLE

The selected DAC channel data format is not signed

LL_DAC_SIGNED_FORMAT_ENABLE

The selected DAC channel data format is signed

DAC trigger source

LL_DAC_TRIG_SOFTWARE

DAC channel conversion trigger internal (SW start)

LL_DAC_TRIG_EXT_TIM1_TRGO

DAC channel conversion trigger from external peripheral: TIM1 TRGO.

LL_DAC_TRIG_EXT_TIM2_TRGO

DAC channel conversion trigger from external peripheral: TIM2 TRGO.

LL_DAC_TRIG_EXT_TIM4_TRGO

DAC channel conversion trigger from external peripheral: TIM4 TRGO.

LL_DAC_TRIG_EXT_TIM5_TRGO

DAC channel conversion trigger from external peripheral: TIM5 TRGO.

LL_DAC_TRIG_EXT_TIM6_TRGO

DAC channel conversion trigger from external peripheral: TIM6 TRGO.

LL_DAC_TRIG_EXT_TIM7_TRGO

DAC channel conversion trigger from external peripheral: TIM7 TRGO.

LL_DAC_TRIG_EXT_TIM8_TRGO

DAC channel conversion trigger from external peripheral: TIM8 TRGO.

LL_DAC_TRIG_EXT_TIM15_TRGO

DAC channel conversion trigger from external peripheral: TIM15 TRGO.

LL_DAC_TRIG_EXT_LPTIM1_CH1

DAC channel conversion trigger from external peripheral: LPTIM1 CH1.

LL_DAC_TRIG_EXT_LPTIM3_CH1

DAC channel conversion trigger from external peripheral: LPTIM3 CH1.

LL_DAC_TRIG_EXT_EXTI_LINE9

DAC channel conversion trigger from external peripheral: external interrupt line 9.

LL_DAC_TRIG_EXT_LPTIM1_OUT

Keep old definition for compatibility

LL_DAC_TRIG_EXT_LPTIM3_OUT

Keep old definition for compatibility

DAC waveform automatic generation mode

LL_DAC_WAVE_AUTO_GENERATION_NONE

DAC channel wave auto generation mode disabled.

LL_DAC_WAVE_AUTO_GENERATION_NOISE

DAC channel wave auto generation mode enabled, set generated noise waveform.

LL_DAC_WAVE_AUTO_GENERATION_TRIANGLE

DAC channel wave auto generation mode enabled, set generated triangle waveform.

DAC wave generation - Noise LFSR unmask bits

LL_DAC_NOISE_LFSR_UNMASK_BIT0

Noise wave generation, unmask LFSR bit0, for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS1_0

Noise wave generation, unmask LFSR bits[1:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS2_0

Noise wave generation, unmask LFSR bits[2:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS3_0

Noise wave generation, unmask LFSR bits[3:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS4_0

Noise wave generation, unmask LFSR bits[4:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS5_0

Noise wave generation, unmask LFSR bits[5:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS6_0

Noise wave generation, unmask LFSR bits[6:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS7_0

Noise wave generation, unmask LFSR bits[7:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS8_0

Noise wave generation, unmask LFSR bits[8:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS9_0

Noise wave generation, unmask LFSR bits[9:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS10_0

Noise wave generation, unmask LFSR bits[10:0], for the selected DAC channel

LL_DAC_NOISE_LFSR_UNMASK_BITS11_0

Noise wave generation, unmask LFSR bits[11:0], for the selected DAC channel

DAC wave generation - Triangle amplitude

LL_DAC_TRIANGLE_AMPLITUDE_1

Triangle wave generation, amplitude of 1 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_3

Triangle wave generation, amplitude of 3 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_7

Triangle wave generation, amplitude of 7 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_15

Triangle wave generation, amplitude of 15 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_31

Triangle wave generation, amplitude of 31 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_63

Triangle wave generation, amplitude of 63 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_127

Triangle wave generation, amplitude of 127 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_255

Triangle wave generation, amplitude of 255 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_511

Triangle wave generation, amplitude of 512 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_1023

Triangle wave generation, amplitude of 1023 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_2047

Triangle wave generation, amplitude of 2047 LSB of DAC output range, for the selected DAC channel

LL_DAC_TRIANGLE_AMPLITUDE_4095

Triangle wave generation, amplitude of 4095 LSB of DAC output range, for the selected DAC channel

DAC helper macro

__LL_DAC_CHANNEL_TO_DECIMAL_NB

Description:

- Helper macro to get DAC channel number in decimal format from literals LL_DAC_CHANNEL_x.

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Return value:

- 1...2

Notes:

- The input can be a value from functions where a channel number is returned.

__LL_DAC_DECIMAL_NB_TO_CHANNEL

Description:

- Helper macro to get DAC channel in literal format LL_DAC_CHANNEL_x from number in decimal format.

Parameters:

- __DECIMAL_NB__: 1...2

Return value:

- Returned: value can be one of the following values:
  – LL_DAC_CHANNEL_1
  – LL_DAC_CHANNEL_2

Notes:

- If the input parameter does not correspond to a DAC channel, this macro returns value '0'.

__LL_DAC_DIGITAL_SCALE

Description:

- Helper macro to define the DAC conversion data full-scale digital value corresponding to the selected DAC resolution.

Parameters:

- __DAC_RESOLUTION__: This parameter can be one of the following values:
  – LL_DAC_RESOLUTION_12B
  – LL_DAC_RESOLUTION_8B

Return value:

- ADC: conversion data equivalent voltage value (unit: mVolt)

Notes:

- DAC conversion data full-scale corresponds to voltage range determined by analog voltage references Vref+ and Vref- (refer to reference manual).

__LL_DAC_CALC_VOLTAGE_TO_DATA

Description:

- Helper macro to calculate the DAC conversion data (unit: digital value) corresponding to a voltage (unit: mVolt).

Parameters:

- __VREFANALOG_VOLTAGE__: Analog reference voltage (unit: mV)
- __DAC_VOLTAGE__: Voltage to be generated by DAC channel (unit: mVolt).
- __DAC_RESOLUTION__: This parameter can be one of the following values:
  – LL_DAC_RESOLUTION_12B
  – LL_DAC_RESOLUTION_8B

Return value:

- DAC: conversion data (unit: digital value)

Notes:

- This helper macro is intended to provide input data in voltage rather than digital value, to be used with LL DAC functions such as LL_DAC_ConvertData12RightAligned(). Analog reference voltage (Vref+) must be either known from user board environment or can be calculated using ADC measurement and ADC helper macro __LL_ADC_CALC_VREFANALOG_VOLTAGE().

Common write and read registers macros

LL_DAC_WriteReg

Description:

- Write a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DAC_ReadReg

Description:

- Read a value in DAC register.

Parameters:

- __INSTANCE__: DAC Instance
- __REG__: Register to be read

Return value:

- Register: value

98 LL DCACHE Generic Driver

98.1 DCACHE Firmware driver API description

The following section lists the various functions of the DCACHE library.

98.1.1 Detailed description of functions

LL_DCACHE_Enable

Function name

__STATIC_INLINE void LL_DCACHE_Enable (DCACHE_TypeDef * DCACHEx)

Function description

Enable the selected DCACHE instance.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_DCACHE_Enable

LL_DCACHE_Disable

Function name

__STATIC_INLINE void LL_DCACHE_Disable (DCACHE_TypeDef * DCACHEx)

Function description

Disable the selected DCACHE instance.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_DCACHE_Disable

LL_DCACHE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabled (const DCACHE_TypeDef * DCACHEx)

Function description

Get the selected DCACHE instance enable state.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **0:** DCACHE is disabled, 1: DCACHE is enabled.

Reference Manual to LL API cross reference:

- CR EN LL_DCACHE_IsEnabled

LL_DCACHE_SetStartAddress

Function name

__STATIC_INLINE void LL_DCACHE_SetStartAddress (DCACHE_TypeDef * DCACHEx, uint32_t addr)

Function description

Set the dcache instance start command address.

Parameters

- **addr:** dcache command start address(Clean, Invalidate or Clean and Invalidate).
- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CMDRSADDRR LL_DCACHE_SetStartAddress

LL_DCACHE_GetStartAddress

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetStartAddress (const DCACHE_TypeDef * DCACHEx)

Function description

Get the dcache command start address.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Start:** address of dcache command

Reference Manual to LL API cross reference:

- CR CMDRSADDRR LL_DCACHE_GetStartAddress

LL_DCACHE_SetEndAddress

Function name

__STATIC_INLINE void LL_DCACHE_SetEndAddress (DCACHE_TypeDef * DCACHEx, uint32_t addr)

Function description

Set the dcache instance End command address.

Parameters

- **DCACHEx:** DCACHE instance
- **addr:** dcache command end address(Clean, Invalidate or Clean and Invalidate).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CMDREADDRR LL_DCACHE_SetEndAddress

LL_DCACHE_GetEndAddress

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetEndAddress (const DCACHE_TypeDef * DCACHEx)

Function description

Get the dcache command End address.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **End:** address of dcache command

Reference Manual to LL API cross reference:

- CR CMDREADDRR LL_DCACHE_GetEndAddress

LL_DCACHE_SetCommand

Function name

__STATIC_INLINE void LL_DCACHE_SetCommand (DCACHE_TypeDef * DCACHEx, uint32_t Command)

Function description

Set Dcache command.

Parameters

- **DCACHEx:** DCACHE instance
- **Command:** command to be applied for the dcache Command can be one of the following values:
  - LL_DCACHE_COMMAND_INVALIDATE_BY_ADDR
  - LL_DCACHE_COMMAND_CLEAN_BY_ADDR
  - LL_DCACHE_COMMAND_CLEAN_INVALIDATE_BY_ADDR
  - LL_DCACHE_COMMAND_NO_OPERATION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CACHECMD LL_DCACHE_SetCommand

LL_DCACHE_GetCommand

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetCommand (const DCACHE_TypeDef * DCACHEx)

Function description

Set Dcache command.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DCACHE_COMMAND_NO_OPERATION
  - LL_DCACHE_COMMAND_CLEAN_BY_ADDR
  - LL_DCACHE_COMMAND_INVALIDATE_BY_ADDR
  - LL_DCACHE_COMMAND_CLEAN_INVALIDATE_BY_ADDR

Reference Manual to LL API cross reference:

- CR CACHECMD LL_DCACHE_GetCommand

LL_DCACHE_StartCommand

Function name

__STATIC_INLINE void LL_DCACHE_StartCommand (DCACHE_TypeDef * DCACHEx)

Function description

Launch Dcache Command.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CACHECMD LL_DCACHE_StartCommand

LL_DCACHE_SetReadBurstType

Function name

__STATIC_INLINE void LL_DCACHE_SetReadBurstType (DCACHE_TypeDef * DCACHEx, uint32_t ReadBurstType)

Function description

Set requested read burst type.

Parameters

- **DCACHEx:** DCACHE instance
- **ReadBurstType:** Burst type to be applied for Data Cache Burst type can be one of the following values:
  - LL_DCACHE_READ_BURST_WRAP
  - LL_DCACHE_READ_BURST_INCR

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HBURST LL_DCACHE_SetReadBurstType

LL_DCACHE_GetReadBurstType

Function name

__STATIC_INLINE uint32_t LL_DCACHE_GetReadBurstType (const DCACHE_TypeDef * DCACHEx)

Function description

Get requested read burst type.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DCACHE_READ_BURST_WRAP
  - LL_DCACHE_READ_BURST_INCR

Reference Manual to LL API cross reference:

- CR HBURST LL_DCACHE_GetReadBurstType

LL_DCACHE_Invalidate

Function name

__STATIC_INLINE void LL_DCACHE_Invalidate (DCACHE_TypeDef * DCACHEx)

Function description

Invalidate the Data cache.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CACHEINV LL_DCACHE_Invalidate

LL_DCACHE_EnableMonitors

Function name

__STATIC_INLINE void LL_DCACHE_EnableMonitors (DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Enable the hit/miss monitor(s).

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** This parameter can be one or a combination of the following values:
  – LL_DCACHE_MONITOR_READ_HIT
  – LL_DCACHE_MONITOR_READ_MISS
  – LL_DCACHE_MONITOR_WRITE_HIT
  – LL_DCACHE_MONITOR_WRITE_MISS
  – LL_DCACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN) LL_DCACHE_EnableMonitors

LL_DCACHE_DisableMonitors

Function name

__STATIC_INLINE void LL_DCACHE_DisableMonitors (DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Disable the hit/miss monitor(s).

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** This parameter can be one or a combination of the following values:
  – LL_DCACHE_MONITOR_READ_HIT
  – LL_DCACHE_MONITOR_READ_MISS
  – LL_DCACHE_MONITOR_WRITE_HIT
  – LL_DCACHE_MONITOR_WRITE_MISS
  – LL_DCACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN) LL_DCACHE_DisableMonitors

LL_DCACHE_IsEnabledMonitors

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledMonitors (const DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Return the hit/miss monitor(s) enable state.

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** This parameter can be one or a combination of the following values:
  – LL_DCACHE_MONITOR_READ_HIT
  – LL_DCACHE_MONITOR_READ_MISS
  – LL_DCACHE_MONITOR_WRITE_HIT
  – LL_DCACHE_MONITOR_WRITE_MISS
  – LL_DCACHE_MONITOR_ALL

Return values

- **State:** of parameter value (1 or 0).

Reference Manual to LL API cross reference:

- CR (RMISSMEN/RHITMEN/WMISSMEN/WHITMEN) LL_DCACHE_IsEnabledMonitors

LL_DCACHE_ResetMonitors

Function name

__STATIC_INLINE void LL_DCACHE_ResetMonitors (DCACHE_TypeDef * DCACHEx, uint32_t Monitors)

Function description

Reset the Data Cache performance monitoring.

Parameters

- **DCACHEx:** DCACHE instance
- **Monitors:** Monitoring type This parameter can be a combination of the following values:
  – LL_DCACHE_MONITOR_READ_HIT
  – LL_DCACHE_MONITOR_READ_MISS
  – LL_DCACHE_MONITOR_WRITE_HIT
  – LL_DCACHE_MONITOR_WRITE_MISS
  – LL_DCACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR (RHITMRST/RMISSMRST/WHITMRST/WMISSMRST) LL_DCACHE_ResetMonitors

LL_DCACHE_Monitor_GetReadHitValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetReadHitValue (const DCACHE_TypeDef * DCACHEx)

Function description

Get the Read Hit monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RHMONR RHITMON LL_DCACHE_Monitor_GetReadHitValue

LL_DCACHE_Monitor_GetReadMissValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetReadMissValue (const DCACHE_TypeDef * DCACHEx)

Function description

Get the Read Miss monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RMMONR RMISSMON LL_DCACHE_Monitor_GetReadMissValue

LL_DCACHE_Monitor_GetWriteHitValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetWriteHitValue (const DCACHE_TypeDef * DCACHEx)

Function description

Get the Write Hit monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- WHMONR WHITMON LL_DCACHE_Monitor_GetWriteHitValue

LL_DCACHE_Monitor_GetWriteMissValue

Function name

__STATIC_INLINE uint32_t LL_DCACHE_Monitor_GetWriteMissValue (const DCACHE_TypeDef * DCACHEx)

Function description

Get the Write Miss monitor Value.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- WMMONR WMISSMON LL_DCACHE_Monitor_GetWriteMissValue

LL_DCACHE_EnableIT_BSYEND

Function name

__STATIC_INLINE void LL_DCACHE_EnableIT_BSYEND (DCACHE_TypeDef * DCACHEx)

Function description

Enable BusyEnd interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_DCACHE_EnableIT_BSYEND

LL_DCACHE_DisableIT_BSYEND

Function name

__STATIC_INLINE void LL_DCACHE_DisableIT_BSYEND (DCACHE_TypeDef * DCACHEx)

Function description

Disable BusyEnd interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_DCACHE_DisableIT_BSYEND

LL_DCACHE_IsEnabledIT_BSYEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledIT_BSYEND (const DCACHE_TypeDef * DCACHEx)

Function description

Indicates whether the Busyend interrupt is enabled.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_DCACHE_IsEnabledIT_BSYEND

LL_DCACHE_EnableIT_ERR

Function name

__STATIC_INLINE void LL_DCACHE_EnableIT_ERR (DCACHE_TypeDef * DCACHEx)

Function description

Enable Error interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_DCACHE_EnableIT_ERR

LL_DCACHE_DisableIT_ERR

Function name

__STATIC_INLINE void LL_DCACHE_DisableIT_ERR (DCACHE_TypeDef * DCACHEx)

Function description

Disable Error interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_DCACHE_DisableIT_ERR

LL_DCACHE_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledIT_ERR (const DCACHE_TypeDef * DCACHEx)

Function description

Indicates whether the Error interrupt is enabled.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_DCACHE_IsEnabledIT_ERR

LL_DCACHE_EnableIT_CMDEND

Function name

__STATIC_INLINE void LL_DCACHE_EnableIT_CMDEND (DCACHE_TypeDef * DCACHEx)

Function description

Enable command end interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_DCACHE_EnableIT_CMDEND

LL_DCACHE_DisableIT_CMDEND

Function name

__STATIC_INLINE void LL_DCACHE_DisableIT_CMDEND (DCACHE_TypeDef * DCACHEx)

Function description

Disable command end interrupt.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_DCACHE_DisableIT_CMDEND

LL_DCACHE_IsEnabledIT_CMDEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsEnabledIT_CMDEND (const DCACHE_TypeDef * DCACHEx)

Function description

Indicates whether the command end interrupt is enabled.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMDENDIE LL_DCACHE_IsEnabledIT_CMDEND

LL_DCACHE_ClearFlag_BSYEND

Function name

__STATIC_INLINE void LL_DCACHE_ClearFlag_BSYEND (DCACHE_TypeDef * DCACHEx)

Function description

Clear full invalidate busy end flag.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CBSYENDF LL_DCACHE_ClearFlag_BSYEND

LL_DCACHE_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_DCACHE_ClearFlag_ERR (DCACHE_TypeDef * DCACHEx)

Function description

Clear cache error flag.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CERRF LL_DCACHE_ClearFlag_ERR

LL_DCACHE_ClearFlag_CMDEND

Function name

__STATIC_INLINE void LL_DCACHE_ClearFlag_CMDEND (DCACHE_TypeDef * DCACHEx)

Function description

Clear command end flag.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CCMDENDF LL_DCACHE_ClearFlag_CMDEND

LL_DCACHE_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_BUSY (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache BUSY.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYF LL_DCACHE_IsActiveFlag_BUSY

LL_DCACHE_IsActiveFlag_BSYEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_BSYEND (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache Busyend.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSYENDF LL_DCACHE_IsActiveFlag_BSYEND

LL_DCACHE_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_ERR (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache Error.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ERRF LL_DCACHE_IsActiveFlag_ERR

LL_DCACHE_IsActiveFlag_BUSYCMD

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_BUSYCMD (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache Busy command.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYCMDF LL_DCACHE_IsActiveFlag_BUSYCMD

LL_DCACHE_IsActiveFlag_CMDEND

Function name

__STATIC_INLINE uint32_t LL_DCACHE_IsActiveFlag_CMDEND (const DCACHE_TypeDef * DCACHEx)

Function description

Get flag Dcache command end.

Parameters

- **DCACHEx:** DCACHE instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CMDENDF LL_DCACHE_IsActiveFlag_CMDEND

98.2 DCACHE Firmware driver defines

The following section lists the various define and macros of the module.

98.2.1 DCACHE

DCACHE

Clear Flags Defines

LL_DCACHE_FCR_CERRF

Cache error flag

LL_DCACHE_FCR_CBSYENDF

Full invalidate busy end flag

LL_DCACHE_FCR_CCMDENDF

Command end flag

Get Flags Defines

LL_DCACHE_SR_ERRF

Cache error flag

LL_DCACHE_SR_BUSYF

Busy flag

LL_DCACHE_SR_CMDENDF

Command end flag

LL_DCACHE_SR_BSYENDF

Full invalidate busy end flag

LL_DCACHE_SR_BUSYCMDF

Command busy flag

IT Defines

LL_DCACHE_IER_BSYENDIE

Busy end interrupt

LL_DCACHE_IER_ERRIE

Cache error interrupt

LL_DCACHE_IER_CMDENDIE

Command end interrupt

Common write and read registers Macros

LL_DCACHE_WriteReg

Description:
- Write a value in DCACHE register.

Parameters:
- __INSTANCE__: DCACHE Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_DCACHE_ReadReg

Description:
- Read a value in DCACHE register.

Parameters:
- __INSTANCE__: DCACHE Instance
- __REG__: Register to be read

Return value:
- Register: value

99 LL DLYB Generic Driver

99.1 DLYB Firmware driver registers structures

99.1.1 LL_DLYB_CfgTypeDef

LL_DLYB_CfgTypeDef is defined in the stm32u5xx_ll_dlyb.h

Data Fields

- ***uint32_t Units***
- ***uint32_t PhaseSel***

Field Documentation

- ***uint32_t LL_DLYB_CfgTypeDef::Units***
  Specifies the Delay of a unit delay cell. This parameter can be a value between 0 and DLYB_MAX_UNIT
- ***uint32_t LL_DLYB_CfgTypeDef::PhaseSel***
  Specifies the Phase for the output clock. This parameter can be a value between 0 and DLYB_MAX_SELECT

99.2 DLYB Firmware driver API description

The following section lists the various functions of the DLYB library.

99.2.1 DelayBlock peripheral features

The DelayBlock is used to generate an Output clock which is de-phased from the Input clock. The phase of the Output clock is programmed by FW. The Output clock is then used to clock the receive data in i.e. a SDMMC, OSPI or QSPI interface. The delay is Voltage and Temperature dependent, which may require FW to do re-tuning and recenter the Output clock phase to the receive data.

The DelayBlock features include the following:

- Input clock frequency.
- Up to 12 oversampling phases.

99.2.2 How to use this driver

This driver is a considered as a driver of service for external devices drivers that interfaces with the DELAY peripheral. The LL_DLYB_SetDelay() function, configure the Delay value configured on SEL and UNIT. The LL_DLYB_GetDelay() function, return the Delay value configured on SEL and UNIT. The LL_DLYB_GetClockPeriod()function, get the clock period.

99.2.3 Detailed description of functions

LL_DLYB_Enable

Function name

__STATIC_INLINE void LL_DLYB_Enable (DLYB_TypeDef * DLYBx)

Function description

DLYB Enable.

Parameters

- **DLYBx:** DLYB Instance

Return values

- **None:**

LL_DLYB_Disable

Function name

__STATIC_INLINE void LL_DLYB_Disable (DLYB_TypeDef * DLYBx)

Function description

Disable the DLYB.

Parameters

- **DLYBx:** DLYB Instance.

Return values

- **None:**

LL_DLYB_SetDelay

Function name

void LL_DLYB_SetDelay (DLYB_TypeDef * DLYBx, LL_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

Set the Delay value configured on SEL and UNIT.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: the Delay value is set.
  – ERROR: the Delay value is not set.

LL_DLYB_GetDelay

Function name

void LL_DLYB_GetDelay (DLYB_TypeDef * DLYBx, LL_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

Get the Delay value configured on SEL and UNIT.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: the Delay value is received.
  – ERROR: the Delay value is not received.

LL_DLYB_GetClockPeriod

Function name

uint32_t LL_DLYB_GetClockPeriod (DLYB_TypeDef * DLYBx, LL_DLYB_CfgTypeDef * pdlyb_cfg)

Function description

Get the clock period.

Parameters

- **DLYBx:** Pointer to DLYB instance.
- **pdlyb_cfg:** Pointer to DLYB configuration structure.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: there is a valid period detected and stored in pdlyb_cfg.
  – ERROR: there is no valid period detected.

99.3 DLYB Firmware driver defines

The following section lists the various define and macros of the module.

99.3.1 DLYB

DLYB

DLYB Flags

DLYB_FLAG_LNGF

100 LL DMA2D Generic Driver

100.1 DMA2D Firmware driver registers structures

100.1.1 LL_DMA2D_InitTypeDef

LL_DMA2D_InitTypeDef is defined in the stm32u5xx_ll_dma2d.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***
- ***uint32_t OutputMemoryAddress***
- ***uint32_t OutputSwapMode***
- ***uint32_t LineOffsetMode***
- ***uint32_t LineOffset***
- ***uint32_t NbrOfLines***
- ***uint32_t NbrOfPixelsPerLines***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***

Field Documentation

- ***uint32_t LL_DMA2D_InitTypeDef::Mode***
  Specifies the DMA2D transfer mode.
  – This parameter can be one value of ***DMA2D_LL_EC_MODE***.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::ColorMode***
  Specifies the color format of the output image.
  – This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColorMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputBlue***
  Specifies the Blue value of the output image.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

- ***uint32_t LL_DMA2D_InitTypeDef::OutputGreen***
  Specifies the Green value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputRed***
  Specifies the Red value of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputAlpha***
  Specifies the Alpha channel of the output image.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.
  - This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.
  - This parameter is not considered if RGB888 or RGB565 color mode is selected.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputMemoryAddress***
  Specifies the memory address.
  - This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputMemAddr()**.
- ***uint32_t LL_DMA2D_InitTypeDef::OutputSwapMode***
  Specifies the output swap mode color format of the output image.
  - This parameter can be one value of *DMA2D_LL_EC_OUTPUT_SWAP_MODE*.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputSwapMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::LineOffsetMode***
  Specifies the output line offset mode.
  - This parameter can be one value of *DMA2D_LL_EC_LINE_OFFSET_MODE*.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetLineOffsetMode()**.

- ***uint32_t LL_DMA2D_InitTypeDef::LineOffset***
  Specifies the output line offset value.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetLineOffset()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfLines***
  Specifies the number of lines of the area to be transferred.
  – This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFF.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetNbrOfLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::NbrOfPixelsPerLines***
  Specifies the number of pixels per lines of the area to be transferred.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetNbrOfPixelsPerLines()**.
- ***uint32_t LL_DMA2D_InitTypeDef::AlphaInversionMode***
  Specifies the output alpha inversion mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputAlphaInvMode()**.
- ***uint32_t LL_DMA2D_InitTypeDef::RBSwapMode***
  Specifies the output Red Blue swap mode.
  – This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP***.
  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputRBSwapMode()**.

100.1.2 LL_DMA2D_LayerCfgTypeDef

LL_DMA2D_LayerCfgTypeDef is defined in the stm32u5xx_ll_dma2d.h

Data Fields

- ***uint32_t MemoryAddress***
- ***uint32_t LineOffset***
- ***uint32_t ColorMode***
- ***uint32_t CLUTColorMode***
- ***uint32_t CLUTSize***
- ***uint32_t AlphaMode***
- ***uint32_t Alpha***
- ***uint32_t Blue***
- ***uint32_t Green***
- ***uint32_t Red***
- ***uint32_t CLUTMemoryAddress***
- ***uint32_t AlphaInversionMode***
- ***uint32_t RBSwapMode***
- ***uint32_t ChromaSubSampling***

Field Documentation

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::MemoryAddress***
  Specifies the foreground or background memory address.
  – This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetMemAddr()** for foreground layer,
  – **LL_DMA2D_BGND_SetMemAddr()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::LineOffset***
  Specifies the foreground or background line offset value.
  – This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetLineOffset()** for foreground layer,
  – **LL_DMA2D_BGND_SetLineOffset()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ColorMode***
  Specifies the foreground or background color mode.
  – This parameter can be one value of ***DMA2D_LL_EC_INPUT_COLOR_MODE***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetColorMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTColorMode***
  Specifies the foreground or background CLUT color mode.
  – This parameter can be one value of ***DMA2D_LL_EC_CLUT_COLOR_MODE***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTColorMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTColorMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTSize***
  Specifies the foreground or background CLUT size.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTSize()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTSize()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaMode***
  Specifies the foreground or background alpha mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_MODE***.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlphaMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlphaMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Alpha***
  Specifies the foreground or background Alpha value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlpha()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlpha()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Blue***
  Specifies the foreground or background Blue color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetBlueColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetBlueColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Green***
  Specifies the foreground or background Green color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetGreenColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetGreenColor()** for background layer.

- ***uint32_t LL_DMA2D_LayerCfgTypeDef::Red***
Specifies the foreground or background Red color value.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetRedColor()** for foreground layer,
  – **LL_DMA2D_BGND_SetRedColor()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::CLUTMemoryAddress***
Specifies the foreground or background CLUT memory address.
  – This parameter must be a number between: Min_Data = 0x0000 and Max_Data = 0xFFFFFFFF.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetCLUTMemAddr()** for foreground layer,
  – **LL_DMA2D_BGND_SetCLUTMemAddr()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::AlphaInversionMode***
Specifies the foreground or background alpha inversion mode.
  – This parameter can be one value of ***DMA2D_LL_EC_ALPHA_INVERSION***.

  This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetAlphaInvMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetAlphaInvMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::RBSwapMode***
Specifies the foreground or background Red Blue swap mode. This parameter can be one value of ***DMA2D_LL_EC_RED_BLUE_SWAP*** .This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetRBSwapMode()** for foreground layer,
  – **LL_DMA2D_BGND_SetRBSwapMode()** for background layer.
- ***uint32_t LL_DMA2D_LayerCfgTypeDef::ChromaSubSampling***
Configure the chroma sub-sampling mode for the YCbCr color mode This parameter is applicable for foreground layer only. This parameter can be one value of ***DMA2D_LL_CHROMA_SUB_SAMPLING***This parameter can be modified afterwards using unitary functions
  – **LL_DMA2D_FGND_SetChrSubSampling()** for foreground layer.

100.1.3 LL_DMA2D_ColorTypeDef

LL_DMA2D_ColorTypeDef is defined in the stm32u5xx_ll_dma2d.h

Data Fields

- ***uint32_t ColorMode***
- ***uint32_t OutputBlue***
- ***uint32_t OutputGreen***
- ***uint32_t OutputRed***
- ***uint32_t OutputAlpha***

Field Documentation

- ***uint32_t LL_DMA2D_ColorTypeDef::ColorMode***
Specifies the color format of the output image.
  – This parameter can be one value of ***DMA2D_LL_EC_OUTPUT_COLOR_MODE***.

  This parameter can be modified afterwards using unitary function **LL_DMA2D_SetOutputColorMode()**.

- ***uint32_t LL_DMA2D_ColorTypeDef::OutputBlue***
  Specifies the Blue value of the output image.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards using, unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputGreen***
  Specifies the Green value of the output image.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  – This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x3F if RGB565 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputRed***
  Specifies the Red value of the output image.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if RGB888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if RGB565 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x1F if ARGB1555 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.
- ***uint32_t LL_DMA2D_ColorTypeDef::OutputAlpha***
  Specifies the Alpha channel of the output image.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0xFF if ARGB8888 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x01 if ARGB1555 color mode is selected.
  – This parameter must be a number between: Min_Data = 0x00 and Max_Data = 0x0F if ARGB4444 color mode is selected.
  – This parameter is not considered if RGB888 or RGB565 color mode is selected.

  This parameter can be modified afterwards, using unitary function **LL_DMA2D_SetOutputColor()** or configuration function **LL_DMA2D_ConfigOutputColor()**.

100.2 DMA2D Firmware driver API description

The following section lists the various functions of the DMA2D library.

100.2.1 Detailed description of functions

LL_DMA2D_Start

Function name

__STATIC_INLINE void LL_DMA2D_Start (DMA2D_TypeDef * DMA2Dx)

Function description

Start a DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR START LL_DMA2D_Start

LL_DMA2D_IsTransferOngoing

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsTransferOngoing (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if a DMA2D transfer is ongoing.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR START LL_DMA2D_IsTransferOngoing

LL_DMA2D_Suspend

Function name

__STATIC_INLINE void LL_DMA2D_Suspend (DMA2D_TypeDef * DMA2Dx)

Function description

Suspend DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to suspend automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_Suspend

LL_DMA2D_Resume

Function name

__STATIC_INLINE void LL_DMA2D_Resume (DMA2D_TypeDef * DMA2Dx)

Function description

Resume DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to resume automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_Resume

LL_DMA2D_IsSuspended

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsSuspended (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D transfer is suspended.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is suspended.

Reference Manual to LL API cross reference:

- CR SUSP LL_DMA2D_IsSuspended

LL_DMA2D_Abort

Function name

__STATIC_INLINE void LL_DMA2D_Abort (DMA2D_TypeDef * DMA2Dx)

Function description

Abort DMA2D transfer.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Notes

- This API can be used to abort automatic foreground or background CLUT loading.

Reference Manual to LL API cross reference:

- CR ABORT LL_DMA2D_Abort

LL_DMA2D_IsAborted

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsAborted (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D transfer is aborted.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This API can be used to indicate whether or not automatic foreground or background CLUT loading is aborted.

Reference Manual to LL API cross reference:

- CR ABORT LL_DMA2D_IsAborted

LL_DMA2D_SetMode

Function name

__STATIC_INLINE void LL_DMA2D_SetMode (DMA2D_TypeDef * DMA2Dx, uint32_t Mode)

Function description

Set DMA2D mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **Mode:** This parameter can be one of the following values:
  - LL_DMA2D_MODE_M2M
  - LL_DMA2D_MODE_M2M_PFC
  - LL_DMA2D_MODE_M2M_BLEND
  - LL_DMA2D_MODE_R2M
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MODE LL_DMA2D_SetMode

LL_DMA2D_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_MODE_M2M
  - LL_DMA2D_MODE_M2M_PFC
  - LL_DMA2D_MODE_M2M_BLEND
  - LL_DMA2D_MODE_R2M
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG
  - LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

Reference Manual to LL API cross reference:

- CR MODE LL_DMA2D_GetMode

LL_DMA2D_SetOutputColorMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D output color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR CM LL_DMA2D_SetOutputColorMode

LL_DMA2D_GetOutputColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_OUTPUT_MODE_ARGB8888
  - LL_DMA2D_OUTPUT_MODE_RGB888
  - LL_DMA2D_OUTPUT_MODE_RGB565
  - LL_DMA2D_OUTPUT_MODE_ARGB1555
  - LL_DMA2D_OUTPUT_MODE_ARGB4444

Reference Manual to LL API cross reference:

- OPFCCR CM LL_DMA2D_GetOutputColorMode

LL_DMA2D_SetOutputRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D output Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR RBS LL_DMA2D_SetOutputRBSwapMode

LL_DMA2D_GetOutputRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputRBSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- OPFCCR RBS LL_DMA2D_GetOutputRBSwapMode

LL_DMA2D_SetOutputAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D output alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  – LL_DMA2D_ALPHA_REGULAR
  – LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR AI LL_DMA2D_SetOutputAlphaInvMode

LL_DMA2D_GetOutputAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputAlphaInvMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_ALPHA_REGULAR
  – LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- OPFCCR AI LL_DMA2D_GetOutputAlphaInvMode

LL_DMA2D_SetOutputSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t OutputSwapMode)

Function description

Set DMA2D output swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputSwapMode:** This parameter can be one of the following values:
  – LL_DMA2D_SWAP_MODE_REGULAR
  – LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Return values

- **None:**

Reference Manual to LL API cross reference:

- OPFCCR SB LL_DMA2D_SetOutputSwapMode

LL_DMA2D_GetOutputSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D output swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_SWAP_MODE_REGULAR
  - LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Reference Manual to LL API cross reference:

- OPFCCR SB LL_DMA2D_GetOutputSwapMode

LL_DMA2D_SetLineOffsetMode

Function name

__STATIC_INLINE void LL_DMA2D_SetLineOffsetMode (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffsetMode)

Function description

Set DMA2D line offset mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffsetMode:** This parameter can be one of the following values:
  - LL_DMA2D_LINE_OFFSET_PIXELS
  - LL_DMA2D_LINE_OFFSET_BYTES

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR LOM LL_DMA2D_SetLineOffsetMode

LL_DMA2D_GetLineOffsetMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffsetMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line offset mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_LINE_OFFSET_PIXELS
  - LL_DMA2D_LINE_OFFSET_BYTES

Reference Manual to LL API cross reference:

- CR LOM LL_DMA2D_GetLineOffsetMode

LL_DMA2D_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- OOR LO LL_DMA2D_SetLineOffset

LL_DMA2D_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineOffset (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Line:** offset value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- OOR LO LL_DMA2D_GetLineOffset

LL_DMA2D_SetNbrOfPixelsPerLines

Function name

__STATIC_INLINE void LL_DMA2D_SetNbrOfPixelsPerLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfPixelsPerLines)

Function description

Set DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- NLR PL LL_DMA2D_SetNbrOfPixelsPerLines

LL_DMA2D_GetNbrOfPixelsPerLines

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfPixelsPerLines (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D number of pixels per lines, expressed on 14 bits ([13:0] bits)

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Number:** of pixels per lines value between Min_Data=0 and Max_Data=0x3FFF

Reference Manual to LL API cross reference:

- NLR PL LL_DMA2D_GetNbrOfPixelsPerLines

LL_DMA2D_SetNbrOfLines

Function name

__STATIC_INLINE void LL_DMA2D_SetNbrOfLines (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines)

Function description

Set DMA2D number of lines, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- NLR NL LL_DMA2D_SetNbrOfLines

LL_DMA2D_GetNbrOfLines

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetNbrOfLines (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D number of lines, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Number:** of lines value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- NLR NL LL_DMA2D_GetNbrOfLines

LL_DMA2D_SetOutputMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t OutputMemoryAddress)

Function description

Set DMA2D output memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- OMAR MA LL_DMA2D_SetOutputMemAddr

LL_DMA2D_GetOutputMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D output memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Output:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- OMAR MA LL_DMA2D_GetOutputMemAddr

LL_DMA2D_SetOutputColor

Function name

__STATIC_INLINE void LL_DMA2D_SetOutputColor (DMA2D_TypeDef * DMA2Dx, uint32_t OutputColor)

Function description

Set DMA2D output color, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **OutputColor:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Notes

- Output color format depends on output color mode, ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444.
- LL_DMA2D_ConfigOutputColor() API may be used instead if colors values formatting with respect to color mode is not done by the user code.

Reference Manual to LL API cross reference:

- OCOLR BLUE LL_DMA2D_SetOutputColor
- OCOLR GREEN LL_DMA2D_SetOutputColor
- OCOLR RED LL_DMA2D_SetOutputColor
- OCOLR ALPHA LL_DMA2D_SetOutputColor

LL_DMA2D_GetOutputColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetOutputColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D output color, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Output:** color value between Min_Data=0 and Max_Data=0xFFFFFFFF

Notes

- Alpha channel and red, green, blue color values must be retrieved from the returned value based on the output color mode (ARGB8888, RGB888, RGB565, ARGB1555 or ARGB4444) as set by LL_DMA2D_SetOutputColorMode.

Reference Manual to LL API cross reference:

- OCOLR BLUE LL_DMA2D_GetOutputColor
- OCOLR GREEN LL_DMA2D_GetOutputColor
- OCOLR RED LL_DMA2D_GetOutputColor
- OCOLR ALPHA LL_DMA2D_GetOutputColor

LL_DMA2D_SetLineWatermark

Function name

__STATIC_INLINE void LL_DMA2D_SetLineWatermark (DMA2D_TypeDef * DMA2Dx, uint32_t LineWatermark)

Function description

Set DMA2D line watermark, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineWatermark:** Value between Min_Data=0 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- LWR LW LL_DMA2D_SetLineWatermark

LL_DMA2D_GetLineWatermark

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetLineWatermark (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D line watermark, expressed on 16 bits ([15:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Line:** watermark value between Min_Data=0 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- LWR LW LL_DMA2D_GetLineWatermark

LL_DMA2D_SetDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_SetDeadTime (DMA2D_TypeDef * DMA2Dx, uint32_t DeadTime)

Function description

Set DMA2D dead time, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **DeadTime:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR DT LL_DMA2D_SetDeadTime

LL_DMA2D_GetDeadTime

Function name

__STATIC_INLINE uint32_t LL_DMA2D_GetDeadTime (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D dead time, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Dead:** time value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- AMTCR DT LL_DMA2D_GetDeadTime

LL_DMA2D_EnableDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_EnableDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D dead time functionality.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_EnableDeadTime

LL_DMA2D_DisableDeadTime

Function name

__STATIC_INLINE void LL_DMA2D_DisableDeadTime (DMA2D_TypeDef * DMA2Dx)

Function description

Disable DMA2D dead time functionality.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_DisableDeadTime

LL_DMA2D_IsEnabledDeadTime

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledDeadTime (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D dead time functionality is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AMTCR EN LL_DMA2D_IsEnabledDeadTime

LL_DMA2D_FGND_SetMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)

Function description

Set DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGMAR MA LL_DMA2D_FGND_SetMemAddr

LL_DMA2D_FGND_GetMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- FGMAR MA LL_DMA2D_FGND_GetMemAddr

LL_DMA2D_FGND_EnableCLUTLoad

Function name

__STATIC_INLINE void LL_DMA2D_FGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D foreground CLUT loading.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR START LL_DMA2D_FGND_EnableCLUTLoad

LL_DMA2D_FGND_IsEnabledCLUTLoad

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_IsEnabledCLUTLoad (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D foreground CLUT loading is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FGPFCCR START LL_DMA2D_FGND_IsEnabledCLUTLoad

LL_DMA2D_FGND_SetColorMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D foreground color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  – LL_DMA2D_INPUT_MODE_ARGB8888
  – LL_DMA2D_INPUT_MODE_RGB888
  – LL_DMA2D_INPUT_MODE_RGB565
  – LL_DMA2D_INPUT_MODE_ARGB1555
  – LL_DMA2D_INPUT_MODE_ARGB4444
  – LL_DMA2D_INPUT_MODE_L8
  – LL_DMA2D_INPUT_MODE_AL44
  – LL_DMA2D_INPUT_MODE_AL88
  – LL_DMA2D_INPUT_MODE_L4
  – LL_DMA2D_INPUT_MODE_A8
  – LL_DMA2D_INPUT_MODE_A4

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CM LL_DMA2D_FGND_SetColorMode

LL_DMA2D_FGND_GetColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_INPUT_MODE_ARGB8888
  – LL_DMA2D_INPUT_MODE_RGB888
  – LL_DMA2D_INPUT_MODE_RGB565
  – LL_DMA2D_INPUT_MODE_ARGB1555
  – LL_DMA2D_INPUT_MODE_ARGB4444
  – LL_DMA2D_INPUT_MODE_L8
  – LL_DMA2D_INPUT_MODE_AL44
  – LL_DMA2D_INPUT_MODE_AL88
  – LL_DMA2D_INPUT_MODE_L4
  – LL_DMA2D_INPUT_MODE_A8
  – LL_DMA2D_INPUT_MODE_A4

Reference Manual to LL API cross reference:

- FGPFCCR CM LL_DMA2D_FGND_GetColorMode

LL_DMA2D_FGND_SetAlphaMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)

Function description

Set DMA2D foreground alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  – LL_DMA2D_ALPHA_MODE_NO_MODIF
  – LL_DMA2D_ALPHA_MODE_REPLACE
  – LL_DMA2D_ALPHA_MODE_COMBINE

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR AM LL_DMA2D_FGND_SetAlphaMode

LL_DMA2D_FGND_GetAlphaMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_ALPHA_MODE_NO_MODIF
  – LL_DMA2D_ALPHA_MODE_REPLACE
  – LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:

- FGPFCCR AM LL_DMA2D_FGND_GetAlphaMode

LL_DMA2D_FGND_SetAlpha

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)

Function description

Set DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL_DMA2D_FGND_SetAlpha

LL_DMA2D_FGND_GetAlpha

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlpha (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Alpha:** value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGPFCCR ALPHA LL_DMA2D_FGND_GetAlpha

LL_DMA2D_FGND_SetRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D foreground Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  – LL_DMA2D_RB_MODE_REGULAR
  – LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR RBS LL_DMA2D_FGND_SetRBSwapMode

LL_DMA2D_FGND_GetRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRBSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_RB_MODE_REGULAR
  – LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- FGPFCCR RBS LL_DMA2D_FGND_GetRBSwapMode

LL_DMA2D_FGND_SetAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D foreground alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  – LL_DMA2D_ALPHA_REGULAR
  – LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR AI LL_DMA2D_FGND_SetAlphaInvMode

LL_DMA2D_FGND_GetAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetAlphaInvMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_ALPHA_REGULAR
  – LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- FGPFCCR AI LL_DMA2D_FGND_GetAlphaInvMode

LL_DMA2D_FGND_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGOR LO LL_DMA2D_FGND_SetLineOffset

LL_DMA2D_FGND_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetLineOffset (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** line offset value between Min_Data=0 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- FGOR LO LL_DMA2D_FGND_GetLineOffset

LL_DMA2D_FGND_SetColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)

Function description

Set DMA2D foreground color values, expressed on 24 bits ([23:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF
- **Green:** Value between Min_Data=0 and Max_Data=0xFF
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_SetColor
- FGCOLR GREEN LL_DMA2D_FGND_SetColor
- FGCOLR BLUE LL_DMA2D_FGND_SetColor

LL_DMA2D_FGND_SetRedColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)

Function description

Set DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_SetRedColor

LL_DMA2D_FGND_GetRedColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetRedColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Red:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR RED LL_DMA2D_FGND_GetRedColor

LL_DMA2D_FGND_SetGreenColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)

Function description

Set DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR GREEN LL_DMA2D_FGND_SetGreenColor

LL_DMA2D_FGND_GetGreenColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetGreenColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Green:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR GREEN LL_DMA2D_FGND_GetGreenColor

LL_DMA2D_FGND_SetBlueColor

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)

Function description

Set DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCOLR BLUE LL_DMA2D_FGND_SetBlueColor

LL_DMA2D_FGND_GetBlueColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetBlueColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Blue:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGCOLR BLUE LL_DMA2D_FGND_GetBlueColor

LL_DMA2D_FGND_SetCLUTMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)

Function description

Set DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGCMAR MA LL_DMA2D_FGND_SetCLUTMemAddr

LL_DMA2D_FGND_GetCLUTMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- FGCMAR MA LL_DMA2D_FGND_GetCLUTMemAddr

LL_DMA2D_FGND_SetCLUTSize

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)

Function description

Set DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CS LL_DMA2D_FGND_SetCLUTSize

LL_DMA2D_FGND_GetCLUTSize

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTSize (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D foreground CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Foreground:** CLUT size value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- FGPFCCR CS LL_DMA2D_FGND_GetCLUTSize

LL_DMA2D_FGND_SetCLUTColorMode

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)

Function description

Set DMA2D foreground CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CCM LL_DMA2D_FGND_SetCLUTColorMode

LL_DMA2D_FGND_GetCLUTColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetCLUTColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Reference Manual to LL API cross reference:

- FGPFCCR CCM LL_DMA2D_FGND_GetCLUTColorMode

LL_DMA2D_FGND_SetChrSubSampling

Function name

__STATIC_INLINE void LL_DMA2D_FGND_SetChrSubSampling (DMA2D_TypeDef * DMA2Dx, uint32_t ChromaSubSampling)

Function description

Set DMA2D foreground Chroma Sub Sampling (for YCbCr input color mode).

Parameters

- **DMA2Dx:** DMA2D Instance
- **ChromaSubSampling:** This parameter can be one of the following values:
  - LL_DMA2D_CSS_444
  - LL_DMA2D_CSS_422
  - LL_DMA2D_CSS_420

Return values

- **None:**

Reference Manual to LL API cross reference:

- FGPFCCR CSS LL_DMA2D_FGND_SetChrSubSampling

LL_DMA2D_FGND_GetChrSubSampling

Function name

__STATIC_INLINE uint32_t LL_DMA2D_FGND_GetChrSubSampling (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D foreground Chroma Sub Sampling (for YCbCr input color mode).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_CSS_444
  – LL_DMA2D_CSS_422
  – LL_DMA2D_CSS_420

Reference Manual to LL API cross reference:

- FGPFCCR CSS LL_DMA2D_FGND_GetChrSubSampling

LL_DMA2D_BGND_SetMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t MemoryAddress)

Function description

Set DMA2D background memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **MemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGMAR MA LL_DMA2D_BGND_SetMemAddr

LL_DMA2D_BGND_GetMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BGMAR MA LL_DMA2D_BGND_GetMemAddr

LL_DMA2D_BGND_EnableCLUTLoad

Function name

__STATIC_INLINE void LL_DMA2D_BGND_EnableCLUTLoad (DMA2D_TypeDef * DMA2Dx)

Function description

Enable DMA2D background CLUT loading.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR START LL_DMA2D_BGND_EnableCLUTLoad

LL_DMA2D_BGND_IsEnabledCLUTLoad

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_IsEnabledCLUTLoad (const DMA2D_TypeDef * DMA2Dx)

Function description

Indicate if DMA2D background CLUT loading is enabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BGPFCCR START LL_DMA2D_BGND_IsEnabledCLUTLoad

LL_DMA2D_BGND_SetColorMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Set DMA2D background color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **ColorMode:** This parameter can be one of the following values:
  – LL_DMA2D_INPUT_MODE_ARGB8888
  – LL_DMA2D_INPUT_MODE_RGB888
  – LL_DMA2D_INPUT_MODE_RGB565
  – LL_DMA2D_INPUT_MODE_ARGB1555
  – LL_DMA2D_INPUT_MODE_ARGB4444
  – LL_DMA2D_INPUT_MODE_L8
  – LL_DMA2D_INPUT_MODE_AL44
  – LL_DMA2D_INPUT_MODE_AL88
  – LL_DMA2D_INPUT_MODE_L4
  – LL_DMA2D_INPUT_MODE_A8
  – LL_DMA2D_INPUT_MODE_A4

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CM LL_DMA2D_BGND_SetColorMode

LL_DMA2D_BGND_GetColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA2D_INPUT_MODE_ARGB8888
  – LL_DMA2D_INPUT_MODE_RGB888
  – LL_DMA2D_INPUT_MODE_RGB565
  – LL_DMA2D_INPUT_MODE_ARGB1555
  – LL_DMA2D_INPUT_MODE_ARGB4444
  – LL_DMA2D_INPUT_MODE_L8
  – LL_DMA2D_INPUT_MODE_AL44
  – LL_DMA2D_INPUT_MODE_AL88
  – LL_DMA2D_INPUT_MODE_L4
  – LL_DMA2D_INPUT_MODE_A8
  – LL_DMA2D_INPUT_MODE_A4

Reference Manual to LL API cross reference:

- BGPFCCR CM LL_DMA2D_BGND_GetColorMode

LL_DMA2D_BGND_SetAlphaMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaMode (DMA2D_TypeDef * DMA2Dx, uint32_t AphaMode)

Function description

Set DMA2D background alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AphaMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR AM LL_DMA2D_BGND_SetAlphaMode

LL_DMA2D_BGND_GetAlphaMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_MODE_NO_MODIF
  - LL_DMA2D_ALPHA_MODE_REPLACE
  - LL_DMA2D_ALPHA_MODE_COMBINE

Reference Manual to LL API cross reference:

- BGPFCCR AM LL_DMA2D_BGND_GetAlphaMode

LL_DMA2D_BGND_SetAlpha

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlpha (DMA2D_TypeDef * DMA2Dx, uint32_t Alpha)

Function description

Set DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Alpha:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR ALPHA LL_DMA2D_BGND_SetAlpha

LL_DMA2D_BGND_GetAlpha

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlpha (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Alpha:** value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGPFCCR ALPHA LL_DMA2D_BGND_GetAlpha

LL_DMA2D_BGND_SetRBSwapMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetRBSwapMode (DMA2D_TypeDef * DMA2Dx, uint32_t RBSwapMode)

Function description

Set DMA2D background Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **RBSwapMode:** This parameter can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR RBS LL_DMA2D_BGND_SetRBSwapMode

LL_DMA2D_BGND_GetRBSwapMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRBSwapMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background Red Blue swap mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_RB_MODE_REGULAR
  - LL_DMA2D_RB_MODE_SWAP

Reference Manual to LL API cross reference:

- BGPFCCR RBS LL_DMA2D_BGND_GetRBSwapMode

LL_DMA2D_BGND_SetAlphaInvMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetAlphaInvMode (DMA2D_TypeDef * DMA2Dx, uint32_t AlphaInversionMode)

Function description

Set DMA2D background alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **AlphaInversionMode:** This parameter can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR AI LL_DMA2D_BGND_SetAlphaInvMode

LL_DMA2D_BGND_GetAlphaInvMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetAlphaInvMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background alpha inversion mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_ALPHA_REGULAR
  - LL_DMA2D_ALPHA_INVERTED

Reference Manual to LL API cross reference:

- BGPFCCR AI LL_DMA2D_BGND_GetAlphaInvMode

LL_DMA2D_BGND_SetLineOffset

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetLineOffset (DMA2D_TypeDef * DMA2Dx, uint32_t LineOffset)

Function description

Set DMA2D background line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **LineOffset:** Value between Min_Data=0 and Max_Data=0x3FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGOR LO LL_DMA2D_BGND_SetLineOffset

LL_DMA2D_BGND_GetLineOffset

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetLineOffset (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background line offset, expressed on 14 bits ([13:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** line offset value between Min_Data=0 and Max_Data=0x3FF

Reference Manual to LL API cross reference:

- BGOR LO LL_DMA2D_BGND_GetLineOffset

LL_DMA2D_BGND_SetColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red, uint32_t Green, uint32_t Blue)

Function description

Set DMA2D background color values, expressed on 24 bits ([23:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF
- **Green:** Value between Min_Data=0 and Max_Data=0xFF
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_SetColor
- BGCOLR GREEN LL_DMA2D_BGND_SetColor
- BGCOLR BLUE LL_DMA2D_BGND_SetColor

LL_DMA2D_BGND_SetRedColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetRedColor (DMA2D_TypeDef * DMA2Dx, uint32_t Red)

Function description

Set DMA2D background red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Red:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_SetRedColor

LL_DMA2D_BGND_GetRedColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetRedColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background red color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Red:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR RED LL_DMA2D_BGND_GetRedColor

LL_DMA2D_BGND_SetGreenColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetGreenColor (DMA2D_TypeDef * DMA2Dx, uint32_t Green)

Function description

Set DMA2D background green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Green:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR GREEN LL_DMA2D_BGND_SetGreenColor

LL_DMA2D_BGND_GetGreenColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetGreenColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background green color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Green:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR GREEN LL_DMA2D_BGND_GetGreenColor

LL_DMA2D_BGND_SetBlueColor

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetBlueColor (DMA2D_TypeDef * DMA2Dx, uint32_t Blue)

Function description

Set DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **Blue:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCOLR BLUE LL_DMA2D_BGND_SetBlueColor

LL_DMA2D_BGND_GetBlueColor

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetBlueColor (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background blue color value, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Blue:** color value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGCOLR BLUE LL_DMA2D_BGND_GetBlueColor

LL_DMA2D_BGND_SetCLUTMemAddr

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTMemAddr (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTMemoryAddress)

Function description

Set DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTMemoryAddress:** Value between Min_Data=0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGCMAR MA LL_DMA2D_BGND_SetCLUTMemAddr

LL_DMA2D_BGND_GetCLUTMemAddr

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTMemAddr (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background CLUT memory address, expressed on 32 bits ([31:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** CLUT memory address value between Min_Data=0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- BGCMAR MA LL_DMA2D_BGND_GetCLUTMemAddr

LL_DMA2D_BGND_SetCLUTSize

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTSize (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTSize)

Function description

Set DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTSize:** Value between Min_Data=0 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CS LL_DMA2D_BGND_SetCLUTSize

LL_DMA2D_BGND_GetCLUTSize

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTSize (const DMA2D_TypeDef * DMA2Dx)

Function description

Get DMA2D background CLUT size, expressed on 8 bits ([7:0] bits).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Background:** CLUT size value between Min_Data=0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- BGPFCCR CS LL_DMA2D_BGND_GetCLUTSize

LL_DMA2D_BGND_SetCLUTColorMode

Function name

__STATIC_INLINE void LL_DMA2D_BGND_SetCLUTColorMode (DMA2D_TypeDef * DMA2Dx, uint32_t CLUTColorMode)

Function description

Set DMA2D background CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance
- **CLUTColorMode:** This parameter can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Return values

- **None:**

Reference Manual to LL API cross reference:

- BGPFCCR CCM LL_DMA2D_BGND_SetCLUTColorMode

LL_DMA2D_BGND_GetCLUTColorMode

Function name

__STATIC_INLINE uint32_t LL_DMA2D_BGND_GetCLUTColorMode (const DMA2D_TypeDef * DMA2Dx)

Function description

Return DMA2D background CLUT color mode.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA2D_CLUT_COLOR_MODE_ARGB8888
  - LL_DMA2D_CLUT_COLOR_MODE_RGB888

Reference Manual to LL API cross reference:

- BGPFCCR CCM LL_DMA2D_BGND_GetCLUTColorMode

LL_DMA2D_IsActiveFlag_CE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Configuration Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CEIF LL_DMA2D_IsActiveFlag_CE

LL_DMA2D_IsActiveFlag_CTC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CTC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Transfer Complete Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTCIF LL_DMA2D_IsActiveFlag_CTC

LL_DMA2D_IsActiveFlag_CAE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_CAE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Access Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CAEIF LL_DMA2D_IsActiveFlag_CAE

LL_DMA2D_IsActiveFlag_TW

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TW (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Watermark Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TWIF LL_DMA2D_IsActiveFlag_TW

LL_DMA2D_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Complete Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF LL_DMA2D_IsActiveFlag_TC

LL_DMA2D_IsActiveFlag_TE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsActiveFlag_TE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Error Interrupt Flag is set or not.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF LL_DMA2D_IsActiveFlag_TE

LL_DMA2D_ClearFlag_CE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Configuration Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CCEIF LL_DMA2D_ClearFlag_CE

LL_DMA2D_ClearFlag_CTC

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D CLUT Transfer Complete Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CCTCIF LL_DMA2D_ClearFlag_CTC

LL_DMA2D_ClearFlag_CAE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D CLUT Access Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CAECIF LL_DMA2D_ClearFlag_CAE

LL_DMA2D_ClearFlag_TW

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Watermark Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTWIF LL_DMA2D_ClearFlag_TW

LL_DMA2D_ClearFlag_TC

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Complete Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF LL_DMA2D_ClearFlag_TC

LL_DMA2D_ClearFlag_TE

Function name

__STATIC_INLINE void LL_DMA2D_ClearFlag_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Clear DMA2D Transfer Error Interrupt Flag.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF LL_DMA2D_ClearFlag_TE

LL_DMA2D_EnableIT_CE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Configuration Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_EnableIT_CE

LL_DMA2D_EnableIT_CTC

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Enable CLUT Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_EnableIT_CTC

LL_DMA2D_EnableIT_CAE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable CLUT Access Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_EnableIT_CAE

LL_DMA2D_EnableIT_TW

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Watermark Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_EnableIT_TW

LL_DMA2D_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_EnableIT_TC

LL_DMA2D_EnableIT_TE

Function name

__STATIC_INLINE void LL_DMA2D_EnableIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Enable Transfer Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_EnableIT_TE

LL_DMA2D_DisableIT_CE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Configuration Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_DisableIT_CE

LL_DMA2D_DisableIT_CTC

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CTC (DMA2D_TypeDef * DMA2Dx)

Function description

Disable CLUT Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_DisableIT_CTC

LL_DMA2D_DisableIT_CAE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_CAE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable CLUT Access Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_DisableIT_CAE

LL_DMA2D_DisableIT_TW

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TW (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Watermark Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_DisableIT_TW

LL_DMA2D_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TC (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Complete Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_DisableIT_TC

LL_DMA2D_DisableIT_TE

Function name

__STATIC_INLINE void LL_DMA2D_DisableIT_TE (DMA2D_TypeDef * DMA2Dx)

Function description

Disable Transfer Error Interrupt.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_DisableIT_TE

LL_DMA2D_IsEnabledIT_CE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Configuration Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CEIE LL_DMA2D_IsEnabledIT_CE

LL_DMA2D_IsEnabledIT_CTC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CTC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Transfer Complete interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CTCIE LL_DMA2D_IsEnabledIT_CTC

LL_DMA2D_IsEnabledIT_CAE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_CAE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D CLUT Access Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CAEIE LL_DMA2D_IsEnabledIT_CAE

LL_DMA2D_IsEnabledIT_TW

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TW (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Watermark interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TWIE LL_DMA2D_IsEnabledIT_TW

LL_DMA2D_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TC (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Complete interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TCIE LL_DMA2D_IsEnabledIT_TC

LL_DMA2D_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_DMA2D_IsEnabledIT_TE (const DMA2D_TypeDef * DMA2Dx)

Function description

Check if the DMA2D Transfer Error interrupt source is enabled or disabled.

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR TEIE LL_DMA2D_IsEnabledIT_TE

LL_DMA2D_DeInit

Function name

ErrorStatus LL_DMA2D_DeInit (const DMA2D_TypeDef * DMA2Dx)

Function description

De-initialize DMA2D registers (registers restored to their default values).

Parameters

- **DMA2Dx:** DMA2D Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA2D registers are de-initialized
  - ERROR: DMA2D registers are not de-initialized

LL_DMA2D_Init

Function name

ErrorStatus LL_DMA2D_Init (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_InitTypeDef * DMA2D_InitStruct)

Function description

Initialize DMA2D registers according to the specified parameters in DMA2D_InitStruct.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure that contains the configuration information for the specified DMA2D peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: DMA2D registers are initialized according to DMA2D_InitStruct content
  – ERROR: Issue occurred during DMA2D registers initialization

Notes

- DMA2D transfers must be disabled to set initialization bits in configuration registers, otherwise ERROR result is returned.

LL_DMA2D_StructInit

Function name

void LL_DMA2D_StructInit (LL_DMA2D_InitTypeDef * DMA2D_InitStruct)

Function description

Set each LL_DMA2D_InitTypeDef field to default value.

Parameters

- **DMA2D_InitStruct:** pointer to a LL_DMA2D_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_DMA2D_ConfigLayer

Function name

void LL_DMA2D_ConfigLayer (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg, uint32_t LayerIdx)

Function description

Configure the foreground or background according to the specified parameters in the LL_DMA2D_LayerCfgTypeDef structure.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure that contains the configuration information for the specified layer.
- **LayerIdx:** DMA2D Layer index. This parameter can be one of the following values: 0(background) / 1(foreground)

Return values

- **None:**

LL_DMA2D_LayerCfgStructInit

Function name

void LL_DMA2D_LayerCfgStructInit (LL_DMA2D_LayerCfgTypeDef * DMA2D_LayerCfg)

Function description

Set each LL_DMA2D_LayerCfgTypeDef field to default value.

Parameters

- **DMA2D_LayerCfg:** pointer to a LL_DMA2D_LayerCfgTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_DMA2D_ConfigOutputColor

Function name

void LL_DMA2D_ConfigOutputColor (DMA2D_TypeDef * DMA2Dx, LL_DMA2D_ColorTypeDef * DMA2D_ColorStruct)

Function description

Initialize DMA2D output color register according to the specified parameters in DMA2D_ColorStruct.

Parameters

- **DMA2Dx:** DMA2D Instance
- **DMA2D_ColorStruct:** pointer to a LL_DMA2D_ColorTypeDef structure that contains the color configuration information for the specified DMA2D peripheral.

Return values

- **None:**

LL_DMA2D_GetOutputBlueColor

Function name

uint32_t LL_DMA2D_GetOutputBlueColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Blue color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  – LL_DMA2D_OUTPUT_MODE_ARGB8888
  – LL_DMA2D_OUTPUT_MODE_RGB888
  – LL_DMA2D_OUTPUT_MODE_RGB565
  – LL_DMA2D_OUTPUT_MODE_ARGB1555
  – LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Blue color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputGreenColor

Function name

uint32_t LL_DMA2D_GetOutputGreenColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Green color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  – LL_DMA2D_OUTPUT_MODE_ARGB8888
  – LL_DMA2D_OUTPUT_MODE_RGB888
  – LL_DMA2D_OUTPUT_MODE_RGB565
  – LL_DMA2D_OUTPUT_MODE_ARGB1555
  – LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Green color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputRedColor

Function name

uint32_t LL_DMA2D_GetOutputRedColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Red color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  – LL_DMA2D_OUTPUT_MODE_ARGB8888
  – LL_DMA2D_OUTPUT_MODE_RGB888
  – LL_DMA2D_OUTPUT_MODE_RGB565
  – LL_DMA2D_OUTPUT_MODE_ARGB1555
  – LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Red color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_GetOutputAlphaColor

Function name

uint32_t LL_DMA2D_GetOutputAlphaColor (const DMA2D_TypeDef * DMA2Dx, uint32_t ColorMode)

Function description

Return DMA2D output Alpha color.

Parameters

- **DMA2Dx:** DMA2D Instance.
- **ColorMode:** This parameter can be one of the following values:
  – LL_DMA2D_OUTPUT_MODE_ARGB8888
  – LL_DMA2D_OUTPUT_MODE_RGB888
  – LL_DMA2D_OUTPUT_MODE_RGB565
  – LL_DMA2D_OUTPUT_MODE_ARGB1555
  – LL_DMA2D_OUTPUT_MODE_ARGB4444

Return values

- **Output:** Alpha color value between Min_Data=0 and Max_Data=0xFF

LL_DMA2D_ConfigSize

Function name

void LL_DMA2D_ConfigSize (DMA2D_TypeDef * DMA2Dx, uint32_t NbrOfLines, uint32_t NbrOfPixelsPerLines)

Function description

Configure DMA2D transfer size.

Parameters

- **DMA2Dx:** DMA2D Instance
- **NbrOfLines:** Value between Min_Data=0 and Max_Data=0xFFFF
- **NbrOfPixelsPerLines:** Value between Min_Data=0 and Max_Data=0x3FFF

Return values

- **None:**

100.3 DMA2D Firmware driver defines

The following section lists the various define and macros of the module.

100.3.1 DMA2D

DMA2D

Chroma Sub Sampling

LL_DMA2D_CSS_444

No chroma sub-sampling 4:4:4

LL_DMA2D_CSS_422

chroma sub-sampling 4:2:2

LL_DMA2D_CSS_420

chroma sub-sampling 4:2:0

Alpha Inversion

LL_DMA2D_ALPHA_REGULAR

Regular alpha

LL_DMA2D_ALPHA_INVERTED

Inverted alpha

Alpha Mode

LL_DMA2D_ALPHA_MODE_NO_MODIF

No modification of the alpha channel value

LL_DMA2D_ALPHA_MODE_REPLACE

Replace original alpha channel value by programmed alpha value

LL_DMA2D_ALPHA_MODE_COMBINE

Replace original alpha channel value by programmed alpha value with, original alpha channel value

CLUT Color Mode

LL_DMA2D_CLUT_COLOR_MODE_ARGB8888

ARGB8888

LL_DMA2D_CLUT_COLOR_MODE_RGB888

RGB888

Get Flags Defines

LL_DMA2D_FLAG_CEIF

Configuration Error Interrupt Flag

LL_DMA2D_FLAG_CTCIF

CLUT Transfer Complete Interrupt Flag

LL_DMA2D_FLAG_CAEIF

CLUT Access Error Interrupt Flag

LL_DMA2D_FLAG_TWIF

Transfer Watermark Interrupt Flag

LL_DMA2D_FLAG_TCIF

Transfer Complete Interrupt Flag

LL_DMA2D_FLAG_TEIF

Transfer Error Interrupt Flag

Input Color Mode

LL_DMA2D_INPUT_MODE_ARGB8888

ARGB8888

LL_DMA2D_INPUT_MODE_RGB888

RGB888

LL_DMA2D_INPUT_MODE_RGB565

RGB565

LL_DMA2D_INPUT_MODE_ARGB1555

ARGB1555

LL_DMA2D_INPUT_MODE_ARGB4444

ARGB4444

LL_DMA2D_INPUT_MODE_L8

L8

LL_DMA2D_INPUT_MODE_AL44

AL44

LL_DMA2D_INPUT_MODE_AL88

AL88

LL_DMA2D_INPUT_MODE_L4

L4

LL_DMA2D_INPUT_MODE_A8

A8

LL_DMA2D_INPUT_MODE_A4

A4

LL_DMA2D_INPUT_MODE_YCBCR

YCbCr

IT Defines

LL_DMA2D_IT_CEIE

Configuration Error Interrupt

LL_DMA2D_IT_CTCIE

CLUT Transfer Complete Interrupt

LL_DMA2D_IT_CAEIE

CLUT Access Error Interrupt

LL_DMA2D_IT_TWIE

Transfer Watermark Interrupt

LL_DMA2D_IT_TCIE

Transfer Complete Interrupt

LL_DMA2D_IT_TEIE

Transfer Error Interrupt

Line Offset Mode

LL_DMA2D_LINE_OFFSET_PIXELS

Line offsets are expressed in pixels

LL_DMA2D_LINE_OFFSET_BYTES

Line offsets are expressed in bytes

Mode

LL_DMA2D_MODE_M2M

DMA2D memory to memory transfer mode

LL_DMA2D_MODE_M2M_PFC

DMA2D memory to memory with pixel format conversion transfer mode

LL_DMA2D_MODE_M2M_BLEND

DMA2D memory to memory with blending transfer mode

LL_DMA2D_MODE_R2M

DMA2D register to memory transfer mode

LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_FG

DMA2D memory to memory with blending transfer mode and fixed color foreground

LL_DMA2D_MODE_M2M_BLEND_FIXED_COLOR_BG

DMA2D memory to memory with blending transfer mode and fixed color background

Output Color Mode

LL_DMA2D_OUTPUT_MODE_ARGB8888

ARGB8888

LL_DMA2D_OUTPUT_MODE_RGB888

RGB888

LL_DMA2D_OUTPUT_MODE_RGB565

RGB565

LL_DMA2D_OUTPUT_MODE_ARGB1555

ARGB1555

LL_DMA2D_OUTPUT_MODE_ARGB4444

ARGB4444

Swap Mode

LL_DMA2D_SWAP_MODE_REGULAR

Regular order

LL_DMA2D_SWAP_MODE_TWO_BY_TWO

Bytes swapped two by two

Red Blue Swap

LL_DMA2D_RB_MODE_REGULAR

RGB or ARGB

LL_DMA2D_RB_MODE_SWAP

BGR or ABGR

Common Write and read registers Macros

LL_DMA2D_WriteReg

Description:
- Write a value in DMA2D register.

Parameters:
- __INSTANCE__: DMA2D Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_DMA2D_ReadReg

Description:
- Read a value in DMA2D register.

Parameters:
- __INSTANCE__: DMA2D Instance
- __REG__: Register to be read

Return value:
- Register: value

101 LL DMA Generic Driver

101.1 DMA Firmware driver registers structures

101.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32u5xx_ll_dma.h

Data Fields

- ***uint32_t SrcAddress***
- ***uint32_t DestAddress***
- ***uint32_t Direction***
- ***uint32_t BlkHWRequest***
- ***uint32_t DataAlignment***
- ***uint32_t SrcBurstLength***
- ***uint32_t DestBurstLength***
- ***uint32_t SrcDataWidth***
- ***uint32_t DestDataWidth***
- ***uint32_t SrcIncMode***
- ***uint32_t DestIncMode***
- ***uint32_t Priority***
- ***uint32_t BlkDataLength***
- ***uint32_t BlkRptCount***
- ***uint32_t TriggerMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***
- ***uint32_t Request***
- ***uint32_t TransferEventMode***
- ***uint32_t DestHWordExchange***
- ***uint32_t DestByteExchange***
- ***uint32_t SrcByteExchange***
- ***uint32_t SrcAllocatedPort***
- ***uint32_t DestAllocatedPort***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t LinkStepMode***
- ***uint32_t SrcAddrUpdateMode***
- ***uint32_t DestAddrUpdateMode***
- ***uint32_t SrcAddrOffset***
- ***uint32_t DestAddrOffset***
- ***uint32_t BlkRptSrcAddrUpdateMode***
- ***uint32_t BlkRptDestAddrUpdateMode***
- ***uint32_t BlkRptSrcAddrOffset***
- ***uint32_t BlkRptDestAddrOffset***
- ***uint32_t LinkedListBaseAddr***
- ***uint32_t LinkedListAddrOffset***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::SrcAddress***
  This field specify the data transfer source address. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddress()**.

- ***uint32_t LL_DMA_InitTypeDef::DestAddress***
  This field specify the data transfer destination address. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddress()**.
- ***uint32_t LL_DMA_InitTypeDef::Direction***
  This field specify the data transfer direction. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_DIRECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkHWRequest***
  This field specify the hardware request unity. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_BLKHW_REQUEST***. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkHWRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::DataAlignment***
  This field specify the transfer data alignment. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DATA_ALIGNMENT***. This feature can be modified afterwards using unitary function **LL_DMA_SetDataAlignment()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcBurstLength***
  This field specify the source burst length of transfer in bytes. Programming this field is not mandatory for LPDMA channels. This parameter must be a value between Min_Data = 1 and Max_Data = 64. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcBurstLength()**.
- ***uint32_t LL_DMA_InitTypeDef::DestBurstLength***
  This field specify the destination burst length of transfer in bytes. Programming this field is not mandatory for LPDMA channels. This parameter must be a value between Min_Data = 1 and Max_Data = 64. This feature can be modified afterwards using unitary function **LL_DMA_SetDestBurstLength()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcDataWidth***
  This field specify the source data width. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_DATA_WIDTH***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcDataWidth()**.
- ***uint32_t LL_DMA_InitTypeDef::DestDataWidth***
  This field specify the destination data width. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_DATA_WIDTH***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestDataWidth()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcIncMode***
  This field specify the source burst increment mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_INCREMENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::DestIncMode***
  This field specify the destination burst increment mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_INCREMENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***
  This field specify the channel priority level. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_PRIORITY_LEVEL***. This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkDataLength***
  This field specify the length of a block transfer in bytes. Programming this field is mandatory for all available DMA channels. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptCount***
  This field specify the number of repetitions of the current block. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value between 1 and 2048 Min_Data = 0 and Max_Data = 0x000007FF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptCount()**.
- ***uint32_t LL_DMA_InitTypeDef::TriggerMode***
  This field specify the trigger mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTriggerMode()**.

- ***uint32_t LL_DMA_InitTypeDef::TriggerPolarity***
  This field specify the trigger event polarity. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_POLARITY***. This feature can be modified afterwards using unitary function **LL_DMA_SetTriggerPolarity()**.
- ***uint32_t LL_DMA_InitTypeDef::TriggerSelection***
  This field specify the trigger event selection. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRIGGER_SELECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetHWTrigger()**.
- ***uint32_t LL_DMA_InitTypeDef::Request***
  This field specify the peripheral request selection. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_REQUEST_SELECTION***. This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::TransferEventMode***
  This field specify the transfer event mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_TRANSFER_EVENT_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetTransferEventMode()**.
- ***uint32_t LL_DMA_InitTypeDef::DestHWordExchange***
  This field specify the destination half word exchange. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_HALFWORD_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestHWordExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::DestByteExchange***
  This field specify the destination byte exchange. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of ***DMA_LL_EC_DEST_BYTE_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestByteExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcByteExchange***
  This field specify the source byte exchange. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of ***DMA_LL_EC_SRC_BYTE_EXCHANGE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcByteExchange()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAllocatedPort***
  This field specify the source allocated port. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of ***DMA_LL_EC_SOURCE_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAllocatedPort***
  This field specify the destination allocated port. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of ***DMA_LL_EC_DESTINATION_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkAllocatedPort***
  This field specify the linked-list allocated port. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of ***DMA_LL_EC_LINKED_LIST_ALLOCATED_PORT***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkAllocatedPort()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkStepMode***
  This field specify the link step mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of ***DMA_LL_EC_LINK_STEP_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkStepMode()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAddrUpdateMode***
  This field specify the source address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_SRC_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::DestAddrUpdateMode***
  This field specify the destination address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of ***DMA_LL_EC_DEST_ADDR_UPDATE_MODE***. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::SrcAddrOffset***
  This field specifies the source address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x00001FFF. This feature can be modified afterwards using unitary function **LL_DMA_SetSrcAddrUpdateValue()**.

- ***uint32_t LL_DMA_InitTypeDef::DestAddrOffset***
  This field specifies the destination address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x00001FFF. This feature can be modified afterwards using unitary function **LL_DMA_SetDestAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptSrcAddrUpdateMode***
  This field specifies the block repeat source address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of *DMA_LL_EC_BLK_RPT_SRC_ADDR_UPDATE_MODE*. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptSrcAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptDestAddrUpdateMode***
  This field specifies the block repeat destination address update mode. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value of *DMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE*. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptDestAddrUpdate()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptSrcAddrOffset***
  This field specifies the block repeat source address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptSrcAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::BlkRptDestAddrOffset***
  This field specifies the block repeat destination address offset. Programming this field is mandatory only for 2D addressing channels. This parameter can be a value Between 0 to 0x0000FFFF. This feature can be modified afterwards using unitary function **LL_DMA_SetBlkRptDestAddrUpdateValue()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkedListBaseAddr***
  This field specify the linked list base address. Programming this field is mandatory for all available DMA channels. This parameter can be a value Between 0 to 0xFFFF0000 (where the 4 first bytes are always forced to 0). This feature can be modified afterwards using unitary function **LL_DMA_SetLinkedListBaseAddr()**.
- ***uint32_t LL_DMA_InitTypeDef::LinkedListAddrOffset***
  Specifies the linked list address offset. Programming this field is mandatory for all available DMA channels. This parameter can be a value Between 0 to 0x0000FFFC. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkedListAddrOffset()**.

101.1.2 LL_DMA_InitLinkedListTypeDef

LL_DMA_InitLinkedListTypeDef is defined in the stm32u5xx_ll_dma.h

Data Fields

- ***uint32_t Priority***
- ***uint32_t LinkStepMode***
- ***uint32_t LinkAllocatedPort***
- ***uint32_t TransferEventMode***

Field Documentation

- ***uint32_t LL_DMA_InitLinkedListTypeDef::Priority***
  This field specify the channel priority level. Programming this field is mandatory for all available DMA channels. This parameter can be a value of *DMA_LL_EC_PRIORITY_LEVEL*. This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::LinkStepMode***
  This field specify the link step mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of *DMA_LL_EC_LINK_STEP_MODE*. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkStepMode()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::LinkAllocatedPort***
  This field specify the linked-list allocated port. Programming this field is not mandatory for LPDMA channels. This parameter can be a value of *DMA_LL_EC_LINKED_LIST_ALLOCATED_PORT*. This feature can be modified afterwards using unitary function **LL_DMA_SetLinkAllocatedPort()**.
- ***uint32_t LL_DMA_InitLinkedListTypeDef::TransferEventMode***
  This field specify the transfer event mode. Programming this field is mandatory for all available DMA channels. This parameter can be a value of *DMA_LL_EC_TRANSFER_EVENT_MODE*. This feature can be modified afterwards using unitary function **LL_DMA_SetTransferEventMode()**.

101.1.3 LL_DMA_InitNodeTypeDef

LL_DMA_InitNodeTypeDef is defined in the stm32u5xx_ll_dma.h

Data Fields

- ***uint32_t DestSecure***
- ***uint32_t DestAllocatedPort***
- ***uint32_t DestHWordExchange***
- ***uint32_t DestByteExchange***
- ***uint32_t DestBurstLength***
- ***uint32_t DestIncMode***
- ***uint32_t DestDataWidth***
- ***uint32_t SrcSecure***
- ***uint32_t SrcAllocatedPort***
- ***uint32_t SrcByteExchange***
- ***uint32_t DataAlignment***
- ***uint32_t SrcBurstLength***
- ***uint32_t SrcIncMode***
- ***uint32_t SrcDataWidth***
- ***uint32_t TransferEventMode***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerSelection***
- ***uint32_t TriggerMode***
- ***uint32_t BlkHWRequest***
- ***uint32_t Direction***
- ***uint32_t Request***
- ***uint32_t BlkRptDestAddrUpdateMode***
- ***uint32_t BlkRptSrcAddrUpdateMode***
- ***uint32_t DestAddrUpdateMode***
- ***uint32_t SrcAddrUpdateMode***
- ***uint32_t BlkRptCount***
- ***uint32_t BlkDataLength***
- ***uint32_t SrcAddress***
- ***uint32_t DestAddress***
- ***uint32_t DestAddrOffset***
- ***uint32_t SrcAddrOffset***
- ***uint32_t BlkRptDestAddrOffset***
- ***uint32_t BlkRptSrcAddrOffset***
- ***uint32_t UpdateRegisters***
- ***uint32_t NodeType***

Field Documentation

- ***uint32_t LL_DMA_InitNodeTypeDef::DestSecure***
  This field specify the destination secure. This parameter can be a value of *DMA_LL_EC_DESTINATION_SECURITY_ATTRIBUTE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAllocatedPort***
  This field specify the destination allocated port. This parameter can be a value of *DMA_LL_EC_DESTINATION_ALLOCATED_PORT*.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestHWordExchange***
  This field specify the destination half word exchange. This parameter can be a value of *DMA_LL_EC_DEST_HALFWORD_EXCHANGE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestByteExchange***
  This field specify the destination byte exchange. This parameter can be a value of *DMA_LL_EC_DEST_BYTE_EXCHANGE*.

- ***uint32_t LL_DMA_InitNodeTypeDef::DestBurstLength***
  This field specify the destination burst length of transfer in bytes. This parameter must be a value between Min_Data = 1 and Max_Data = 64.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestIncMode***
  This field specify the destination increment mode. This parameter can be a value of *DMA_LL_EC_DESTINATION_INCREMENT_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestDataWidth***
  This field specify the destination data width. This parameter can be a value of *DMA_LL_EC_DESTINATION_DATA_WIDTH*.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcSecure***
  This field specify the source secure. This parameter can be a value of *DMA_LL_EC_SOURCE_SECURITY_ATTRIBUTE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAllocatedPort***
  This field specify the source allocated port. This parameter can be a value of *DMA_LL_EC_SOURCE_ALLOCATED_PORT*.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcByteExchange***
  This field specify the source byte exchange. This parameter can be a value of *DMA_LL_EC_SRC_BYTE_EXCHANGE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::DataAlignment***
  This field specify the transfer data alignment. This parameter can be a value of *DMA_LL_EC_DATA_ALIGNMENT*.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcBurstLength***
  This field specify the source burst length of transfer in bytes. This parameter must be a value between Min_Data = 1 and Max_Data = 64.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcIncMode***
  This field specify the source increment mode. This parameter can be a value of *DMA_LL_EC_SOURCE_INCREMENT_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcDataWidth***
  This field specify the source data width. This parameter can be a value of *DMA_LL_EC_SOURCE_DATA_WIDTH*.
- ***uint32_t LL_DMA_InitNodeTypeDef::TransferEventMode***
  This field specify the transfer event mode. This parameter can be a value of *DMA_LL_EC_TRANSFER_EVENT_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerPolarity***
  This field specify the trigger event polarity. This parameter can be a value of *DMA_LL_EC_TRIGGER_POLARITY*.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerSelection***
  This field specify the trigger event selection. This parameter can be a value of *DMA_LL_EC_TRIGGER_SELECTION*.
- ***uint32_t LL_DMA_InitNodeTypeDef::TriggerMode***
  This field specify the trigger mode. This parameter can be a value of *DMA_LL_EC_TRIGGER_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkHWRequest***
  This field specify the hardware request unity. This parameter can be a value of *DMA_LL_EC_BLKHW_REQUEST*.
- ***uint32_t LL_DMA_InitNodeTypeDef::Direction***
  This field specify the transfer direction. This parameter can be a value of *DMA_LL_EC_TRANSFER_DIRECTION*.
- ***uint32_t LL_DMA_InitNodeTypeDef::Request***
  This field specify the peripheral request selection. This parameter can be a value of *DMA_LL_EC_REQUEST_SELECTION*.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptDestAddrUpdateMode***
  This field specifies the block repeat destination address update mode. This parameter can be a value of *DMA_LL_EC_BLK_RPT_DEST_ADDR_UPDATE_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptSrcAddrUpdateMode***
  This field specifies the block repeat source address update mode. This parameter can be a value of *DMA_LL_EC_BLK_RPT_SRC_ADDR_UPDATE_MODE*.

- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddrUpdateMode***
  This field specify the Destination address update mode. This parameter can be a value of *DMA_LL_EC_DEST_ADDR_UPDATE_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddrUpdateMode***
  This field specify the Source address update mode. This parameter can be a value of *DMA_LL_EC_SRC_ADDR_UPDATE_MODE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptCount***
  This field specify the number of repetitions of the current block. This parameter can be a value between 1 and 2048 Min_Data = 0 and Max_Data = 0x000007FF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkDataLength***
  This field specify the length of a block transfer in bytes. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddress***
  This field specify the transfer source address. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddress***
  This field specify the transfer destination address. This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::DestAddrOffset***
  This field specifies the destination address offset. This parameter can be a value Between 0 to 0x00001FFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::SrcAddrOffset***
  This field specifies the source address offset. This parameter can be a value Between 0 to 0x00001FFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptDestAddrOffset***
  This field specifies the block repeat destination address offset. This parameter can be a value Between 0 to 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::BlkRptSrcAddrOffset***
  This field specifies the block repeat source address offset. This parameter can be a value Between 0 to 0x0000FFFF.
- ***uint32_t LL_DMA_InitNodeTypeDef::UpdateRegisters***
  Specifies the linked list register update. This parameter can be a value of *DMA_LL_EC_LINKEDLIST_REGISTER_UPDATE*.
- ***uint32_t LL_DMA_InitNodeTypeDef::NodeType***
  Specifies the node type to be created. This parameter can be a value of *DMA_LL_EC_LINKEDLIST_NODE_TYPE*.

101.1.4 LL_DMA_LinkNodeTypeDef

LL_DMA_LinkNodeTypeDef is defined in the stm32u5xx_ll_dma.h

Data Fields

- ***__IO uint32_t LinkRegisters***

Field Documentation

- ***__IO uint32_t LL_DMA_LinkNodeTypeDef::LinkRegisters[8U]***

101.2 DMA Firmware driver API description

The following section lists the various functions of the DMA library.

101.2.1 LL DMA driver acronyms

Acronyms table : == || Acronym || || == || SRC || Source || || DEST || Destination || || ADDR || Address || || ADDRS || Addresses || || INC || Increment / Incremented || || DEC || Decrement / Decremented || || BLK || Block || || RPT || Repeat / Repeated || || TRIG || Trigger || ==

101.2.2 Detailed description of functions

LL_DMA_EnableChannel

Function name

__STATIC_INLINE void LL_DMA_EnableChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable channel.

Parameters

- **DMAx:** DMAx Instance.
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name

__STATIC_INLINE void LL_DMA_DisableChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable channel.

Parameters

- **DMAx:** DMAx Instance.
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if channel is enabled or disabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ResetChannel

Function name

__STATIC_INLINE void LL_DMA_ResetChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Reset channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR RESET LL_DMA_ResetChannel

LL_DMA_SuspendChannel

Function name

__STATIC_INLINE void LL_DMA_SuspendChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Suspend channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_SuspendChannel

LL_DMA_ResumeChannel

Function name

__STATIC_INLINE void LL_DMA_ResumeChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Resume channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_ResumeChannel

LL_DMA_IsSuspendedChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsSuspendedChannel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if channel is suspended.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSP LL_DMA_IsSuspendedChannel

LL_DMA_SetLinkedListBaseAddr

Function name

__STATIC_INLINE void LL_DMA_SetLinkedListBaseAddr (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkedListBaseAddr)

Function description

Set linked-list base address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **LinkedListBaseAddr:** Between 0 to 0xFFFF0000 (where the 4 LSB bytes are always 0)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLBAR LBA LL_DMA_SetLinkedListBaseAddr

LL_DMA_GetLinkedListBaseAddr

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkedListBaseAddr (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked-list base address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Value:** between 0 to 0xFFFF0000 (where the 4 LSB bytes are always 0)

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLBAR LBA LL_DMA_GetLinkedListBaseAddr

LL_DMA_ConfigControl

Function name

__STATIC_INLINE void LL_DMA_ConfigControl (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all parameters linked to channel control.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT or LL_DMA_LOW_PRIORITY_MID_WEIGHT or LL_DMA_LOW_PRIORITY_HIGH_WEIGHT or LL_DMA_HIGH_PRIORITY
  - LL_DMA_LINK_ALLOCATED_PORT0 or LL_DMA_LINK_ALLOCATED_PORT1
  - LL_DMA_LSM_FULL_EXECUTION or LL_DMA_LSM_1LINK_EXECUTION

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For LPDMA channels, LAP field programming is discarded.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_ConfigControl
- CCR LAP LL_DMA_ConfigControl
- CCR LSM LL_DMA_ConfigControl

LL_DMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)

Function description

Set priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Priority:** This parameter can be one of the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT
  - LL_DMA_LOW_PRIORITY_MID_WEIGHT
  - LL_DMA_LOW_PRIORITY_HIGH_WEIGHT
  - LL_DMA_HIGH_PRIORITY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Channel priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_LOW_PRIORITY_LOW_WEIGHT
  - LL_DMA_LOW_PRIORITY_MID_WEIGHT
  - LL_DMA_LOW_PRIORITY_HIGH_WEIGHT
  - LL_DMA_HIGH_PRIORITY

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR PRIO LL_DMA_GetChannelPriorityLevel

LL_DMA_SetLinkAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetLinkAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkAllocatedPort)

Function description

Set linked-list allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **LinkAllocatedPort:** This parameter can be one of the following values:
  – LL_DMA_LINK_ALLOCATED_PORT0
  – LL_DMA_LINK_ALLOCATED_PORT1

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CCR LAP LL_DMA_SetLinkAllocatedPort

LL_DMA_GetLinkAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked-list allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_LINK_ALLOCATED_PORT0
  – LL_DMA_LINK_ALLOCATED_PORT1

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CCR LAP LL_DMA_GetLinkAllocatedPort

LL_DMA_SetLinkStepMode

Function name

__STATIC_INLINE void LL_DMA_SetLinkStepMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkStepMode)

Function description

Set link step mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **LinkStepMode:** This parameter can be one of the following values:
  – LL_DMA_LSM_FULL_EXECUTION
  – LL_DMA_LSM_1LINK_EXECUTION

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR LSM LL_DMA_SetLinkStepMode

LL_DMA_GetLinkStepMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkStepMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Link step mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_LSM_FULL_EXECUTION
  – LL_DMA_LSM_1LINK_EXECUTION

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR LSM LL_DMA_GetLinkStepMode

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure data transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0 or LL_DMA_DEST_ALLOCATED_PORT1
  - LL_DMA_DEST_HALFWORD_PRESERVE or LL_DMA_DEST_HALFWORD_EXCHANGE
  - LL_DMA_DEST_BYTE_PRESERVE or LL_DMA_DEST_BYTE_EXCHANGE
  - LL_DMA_SRC_BYTE_PRESERVE or LL_DMA_SRC_BYTE_EXCHANGE
  - LL_DMA_DEST_FIXED or LL_DMA_DEST_INCREMENT
  - LL_DMA_DEST_DATAWIDTH_BYTE or LL_DMA_DEST_DATAWIDTH_HALFWORD or LL_DMA_DEST_DATAWIDTH_WORD
  - LL_DMA_SRC_ALLOCATED_PORT0 or LL_DMA_SRC_ALLOCATED_PORT1
  - LL_DMA_DATA_ALIGN_ZEROPADD or LL_DMA_DATA_ALIGN_SIGNEXTPADD or LL_DMA_DATA_PACK_UNPACK
  - LL_DMA_SRC_FIXED or LL_DMA_SRC_INCREMENT
  - LL_DMA_SRC_DATAWIDTH_BYTE or LL_DMA_SRC_DATAWIDTH_HALFWORD or LL_DMA_SRC_DATAWIDTH_WORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For LPDMA channels DAP, DHX, DBX, SAP, SBX fields programming is discarded.

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_ConfigTransfer
- CTR1 DHX LL_DMA_ConfigTransfer
- CTR1 DBX LL_DMA_ConfigTransfer
- CTR1 DINC LL_DMA_ConfigTransfer
- CTR1 SAP LL_DMA_ConfigTransfer
- CTR1 SBX LL_DMA_ConfigTransfer
- CTR1 PAM LL_DMA_ConfigTransfer
- CTR1 SINC LL_DMA_ConfigTransfer

LL_DMA_ConfigBurstLength

Function name

__STATIC_INLINE void LL_DMA_ConfigBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcBurstLength, uint32_t DestBurstLength)

Function description

Configure source and destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcBurstLength:** Between 1 to 64
- **DestBurstLength:** Between 1 to 64

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_SetDestBurstLength
-
- CTR1 SBL_1 LL_DMA_SetDestBurstLength

LL_DMA_ConfigChannelSecure

Function name

__STATIC_INLINE void LL_DMA_ConfigChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all secure parameters linked to DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_CHANNEL_NSEC or LL_DMA_CHANNEL_SEC
  - LL_DMA_CHANNEL_SRC_NSEC or LL_DMA_CHANNEL_SRC_SEC
  - LL_DMA_CHANNEL_DEST_NSEC or LL_DMA_CHANNEL_DEST_SEC

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SEC LL_DMA_ConfigChannelSecure
-
- CTR1 SSEC LL_DMA_ConfigChannelSecure
-
- CTR1 DSEC LL_DMA_ConfigChannelSecure

LL_DMA_EnableChannelDestSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_EnableChannelDestSecure

LL_DMA_DisableChannelDestSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_DisableChannelDestSecure

LL_DMA_IsEnabledChannelDestSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelDestSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check security attribute of the DMA transfer to the destination.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DSEC LL_DMA_IsEnabledChannelDestSecure

LL_DMA_EnableChannelSrcSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_EnableChannelSrcSecure

LL_DMA_DisableChannelSrcSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_DisableChannelSrcSecure

LL_DMA_IsEnabledChannelSrcSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelSrcSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check security attribute of the DMA transfer from the source.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SSEC LL_DMA_IsEnabledChannelSrcSecure

LL_DMA_SetDestAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetDestAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAllocatedPort)

Function description

Set destination allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **DestAllocatedPort:** This parameter can be one of the following values:
  – LL_DMA_DEST_ALLOCATED_PORT0
  – LL_DMA_DEST_ALLOCATED_PORT1

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_SetDestAllocatedPort

LL_DMA_GetDestAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_ALLOCATED_PORT0
  - LL_DMA_DEST_ALLOCATED_PORT1

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DAP LL_DMA_GetDestAllocatedPort

LL_DMA_SetDestHWordExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestHWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestHWordExchange)

Function description

Set destination half-word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestHWordExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_HALFWORD_PRESERVE
  - LL_DMA_DEST_HALFWORD_EXCHANGE

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DHX LL_DMA_SetDestHWordExchange

LL_DMA_GetDestHWordExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestHWordExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination half-word exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_HALFWORD_PRESERVE
  - LL_DMA_DEST_HALFWORD_EXCHANGE

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DHX LL_DMA_GetDestHWordExchange

LL_DMA_SetDestByteExchange

Function name

__STATIC_INLINE void LL_DMA_SetDestByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestByteExchange)

Function description

Set destination byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestByteExchange:** This parameter can be one of the following values:
  - LL_DMA_DEST_BYTE_PRESERVE
  - LL_DMA_DEST_BYTE_EXCHANGE

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DBX LL_DMA_SetDestByteExchange

LL_DMA_GetDestByteExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_DEST_BYTE_PRESERVE
  – LL_DMA_DEST_BYTE_EXCHANGE

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DBX LL_DMA_GetDestByteExchange

LL_DMA_SetSrcByteExchange

Function name

__STATIC_INLINE void LL_DMA_SetSrcByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcByteExchange)

Function description

Set source byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **SrcByteExchange:** This parameter can be one of the following values:
  – LL_DMA_SRC_BYTE_PRESERVE
  – LL_DMA_SRC_BYTE_EXCHANGE

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 SBX LL_DMA_SetSrcByteExchange

LL_DMA_GetSrcByteExchange

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcByteExchange (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source byte exchange.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_SRC_BYTE_PRESERVE
  – LL_DMA_SRC_BYTE_EXCHANGE

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 SBX LL_DMA_GetSrcByteExchange

LL_DMA_SetDestBurstLength

Function name

__STATIC_INLINE void LL_DMA_SetDestBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestBurstLength)

Function description

Set destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestBurstLength:** Between 1 to 64

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_SetDestBurstLength

LL_DMA_GetDestBurstLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 1 to 64.

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DBL_1 LL_DMA_GetDestBurstLength

LL_DMA_SetDestIncMode

Function name

__STATIC_INLINE void LL_DMA_SetDestIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestInc)

Function description

Set destination increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestInc:** This parameter can be one of the following values:
  - LL_DMA_DEST_FIXED
  - LL_DMA_DEST_INCREMENT

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 DINC LL_DMA_SetDestIncMode

LL_DMA_GetDestIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_DEST_FIXED
  – LL_DMA_DEST_INCREMENT

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DINC LL_DMA_GetDestIncMode

LL_DMA_SetDestDataWidth

Function name

__STATIC_INLINE void LL_DMA_SetDestDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestDataWidth)

Function description

Set destination data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestDataWidth:** This parameter can be one of the following values:
  - LL_DMA_DEST_DATAWIDTH_BYTE
  - LL_DMA_DEST_DATAWIDTH_HALFWORD
  - LL_DMA_DEST_DATAWIDTH_WORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DDW_LOG2 LL_DMA_SetDestDataWidth

LL_DMA_GetDestDataWidth

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DEST_DATAWIDTH_BYTE
  - LL_DMA_DEST_DATAWIDTH_HALFWORD
  - LL_DMA_DEST_DATAWIDTH_WORD

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 DDW_LOG2 LL_DMA_GetDestDataWidth

LL_DMA_SetSrcAllocatedPort

Function name

__STATIC_INLINE void LL_DMA_SetSrcAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAllocatedPort)

Function description

Set source allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAllocatedPort:** This parameter can be one of the following values:
  - LL_DMA_SRC_ALLOCATED_PORT0
  - LL_DMA_SRC_ALLOCATED_PORT1

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 SAP LL_DMA_SetSrcAllocatedPort

LL_DMA_GetSrcAllocatedPort

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAllocatedPort (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source allocated port.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_SRC_ALLOCATED_PORT0
  - LL_DMA_SRC_ALLOCATED_PORT1

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 SAP LL_DMA_GetSrcAllocatedPort

LL_DMA_SetDataAlignment

Function name

__STATIC_INLINE void LL_DMA_SetDataAlignment (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DataAlignment)

Function description

Set data alignment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **DataAlignment:** This parameter can be one of the following values:
  – LL_DMA_DATA_ALIGN_ZEROPADD
  – LL_DMA_DATA_ALIGN_SIGNEXTPADD
  – LL_DMA_DATA_PACK_UNPACK (This value is not allowed for LPDMA channels)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For LPDMA channels, PAM field is reduced to one bit.

Reference Manual to LL API cross reference:

- CTR1 PAM LL_DMA_SetDataAlignment

LL_DMA_GetDataAlignment

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataAlignment (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get data alignment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_DATA_ALIGN_ZEROPADD
  – LL_DMA_DATA_ALIGN_SIGNEXTPADD
  – LL_DMA_DATA_PACK_UNPACK (This value is not allowed for LPDMA channels)

Notes

- This API is used for all available DMA channels. For LPDMA channels, PAM field is reduced to one bit.

Reference Manual to LL API cross reference:

- CTR1 PAM LL_DMA_GetDataAlignment

LL_DMA_SetSrcBurstLength

Function name

__STATIC_INLINE void LL_DMA_SetSrcBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcBurstLength)

Function description

Set source burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcBurstLength:** Between 1 to 64

Return values

- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 SBL_1 LL_DMA_SetSrcBurstLength

LL_DMA_GetSrcBurstLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcBurstLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source burst length.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 1 to 64
- **None.:**

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CTR1 SBL_1 LL_DMA_GetSrcBurstLength

LL_DMA_SetSrcIncMode

Function name

__STATIC_INLINE void LL_DMA_SetSrcIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcInc)

Function description

Set source increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcInc:** This parameter can be one of the following values:
  - LL_DMA_SRC_FIXED
  - LL_DMA_SRC_INCREMENT

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SINC LL_DMA_SetSrcIncMode

LL_DMA_GetSrcIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_SRC_FIXED
  – LL_DMA_SRC_INCREMENT

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SINC LL_DMA_GetSrcIncMode

LL_DMA_SetSrcDataWidth

Function name

__STATIC_INLINE void LL_DMA_SetSrcDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcDataWidth)

Function description

Set source data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **SrcDataWidth:** This parameter can be one of the following values:
  – LL_DMA_SRC_DATAWIDTH_BYTE
  – LL_DMA_SRC_DATAWIDTH_HALFWORD
  – LL_DMA_SRC_DATAWIDTH_WORD

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SDW_LOG2 LL_DMA_SetSrcDataWidth

LL_DMA_GetSrcDataWidth

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcDataWidth (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Source Data width.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_SRC_DATAWIDTH_BYTE
  – LL_DMA_SRC_DATAWIDTH_HALFWORD
  – LL_DMA_SRC_DATAWIDTH_WORD

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR1 SDW_LOG2 LL_DMA_GetSrcDataWidth

LL_DMA_ConfigChannelTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigChannelTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure channel transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_TCEM_BLK_TRANSFER or LL_DMA_TCEM_RPT_BLK_TRANSFER or LL_DMA_TCEM_EACH_LLITEM_TRANSFER or LL_DMA_TCEM_LAST_LLITEM_TRANSFER
  - LL_DMA_HWREQUEST_SINGLEBURST or LL_DMA_HWREQUEST_BLK
  - LL_DMA_TRIG_POLARITY_MASKED or LL_DMA_TRIG_POLARITY_RISING or LL_DMA_TRIG_POLARITY_FALLING
  - LL_DMA_TRIGM_BLK_TRANSFER or LL_DMA_TRIGM_RPT_BLK_TRANSFER or LL_DMA_TRIGM_LLI_LINK_TRANSFER or LL_DMA_TRIGM_SINGLBURST_TRANSFER
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_ConfigChannelTransfer
- CTR2 TRIGPOL LL_DMA_ConfigChannelTransfer
- CTR2 TRIGM LL_DMA_ConfigChannelTransfer
- CTR2 BREQ LL_DMA_ConfigChannelTransfer
- CTR2 DREQ LL_DMA_ConfigChannelTransfer
- CTR2 SWREQ LL_DMA_ConfigChannelTransfer

LL_DMA_SetTransferEventMode

Function name

__STATIC_INLINE void LL_DMA_SetTransferEventMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TransferEventMode)

Function description

Set transfer event mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **TransferEventMode:** This parameter can be one of the following values:
  - LL_DMA_TCEM_BLK_TRANSFER
  - LL_DMA_TCEM_RPT_BLK_TRANSFER
  - LL_DMA_TCEM_EACH_LLITEM_TRANSFER
  - LL_DMA_TCEM_LAST_LLITEM_TRANSFER

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_SetTransferEventMode

LL_DMA_GetTransferEventMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTransferEventMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get transfer event mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TCEM_BLK_TRANSFER
  - LL_DMA_TCEM_RPT_BLK_TRANSFER
  - LL_DMA_TCEM_EACH_LLITEM_TRANSFER
  - LL_DMA_TCEM_LAST_LLITEM_TRANSFER

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TCEM LL_DMA_GetTransferEventMode

LL_DMA_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_DMA_SetTriggerPolarity (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TriggerPolarity)

Function description

Set trigger polarity.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **TriggerPolarity:** This parameter can be one of the following values:
  - LL_DMA_TRIG_POLARITY_MASKED
  - LL_DMA_TRIG_POLARITY_RISING
  - LL_DMA_TRIG_POLARITY_FALLING

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGPOL LL_DMA_SetTriggerPolarity

LL_DMA_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTriggerPolarity (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger polarity.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TRIG_POLARITY_MASKED
  - LL_DMA_TRIG_POLARITY_RISING
  - LL_DMA_TRIG_POLARITY_FALLING

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGPOL LL_DMA_GetTriggerPolarity

LL_DMA_SetTriggerMode

Function name

__STATIC_INLINE void LL_DMA_SetTriggerMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t TriggerMode)

Function description

Set trigger Mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **TriggerMode:** This parameter can be one of the following values:
  – LL_DMA_TRIGM_BLK_TRANSFER
  – LL_DMA_TRIGM_RPT_BLK_TRANSFER (This value is allowed only for 2D addressing channels)
  – LL_DMA_TRIGM_LLI_LINK_TRANSFER
  – LL_DMA_TRIGM_SINGLBURST_TRANSFER

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGM LL_DMA_SetTriggerMode

LL_DMA_GetTriggerMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetTriggerMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger Mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_TRIGM_BLK_TRANSFER
  - LL_DMA_TRIGM_RPT_BLK_TRANSFER (This value is allowed only for 2D addressing channels)
  - LL_DMA_TRIGM_LLI_LINK_TRANSFER
  - LL_DMA_TRIGM_SINGLBURST_TRANSFER

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 TRIGM LL_DMA_GetTriggerMode

LL_DMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_DMA_SetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description

Set destination hardware and software transfer request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **Direction:** This parameter can be one of the following values:
  – LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  – LL_DMA_DIRECTION_MEMORY_TO_PERIPH (This value is not allowed for LPDMA channels)
  – LL_DMA_DIRECTION_MEMORY_TO_MEMORY (This value is not allowed for LPDMA channels)

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 DREQ LL_DMA_SetDataTransferDirection
-
- CTR2 SWREQ LL_DMA_SetDataTransferDirection

LL_DMA_GetDataTransferDirection

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination hardware and software transfer request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH (This value is not allowed for LPDMA channels)
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY (This value is not allowed for LPDMA channels)

Notes

- This API is used for all available DMA channels. For LPDMA channels, DREQ fields programming is discarded.

Reference Manual to LL API cross reference:

- CTR2 DREQ LL_DMA_GetDataTransferDirection
-
- CTR2 SWREQ LL_DMA_GetDataTransferDirection

LL_DMA_SetBlkHWRequest

Function name

__STATIC_INLINE void LL_DMA_SetBlkHWRequest (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkHWRequest)

Function description

Set block hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkHWRequest:** This parameter can be one of the following values:
  - LL_DMA_HWREQUEST_SINGLEBURST
  - LL_DMA_HWREQUEST_BLK

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 BREQ LL_DMA_SetBlkHWRequest
-

LL_DMA_GetBlkHWRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkHWRequest (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_HWREQUEST_SINGLEBURST
  - LL_DMA_HWREQUEST_BLK

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CTR2 BREQ LL_DMA_GetBlkHWRequest
-

LL_DMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_DMA_SetPeriphRequest (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Request)

Function description

Set hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

- **Request:** This parameter can be one of the following values:
  - LL_GPDMA1_REQUEST_ADC1
  - LL_GPDMA1_REQUEST_ADC4
  - LL_GPDMA1_REQUEST_DAC1_CH1
  - LL_GPDMA1_REQUEST_DAC1_CH2
  - LL_GPDMA1_REQUEST_TIM6_UP
  - LL_GPDMA1_REQUEST_TIM7_UP
  - LL_GPDMA1_REQUEST_SPI1_RX
  - LL_GPDMA1_REQUEST_SPI1_TX
  - LL_GPDMA1_REQUEST_SPI2_RX
  - LL_GPDMA1_REQUEST_SPI2_TX
  - LL_GPDMA1_REQUEST_SPI3_RX
  - LL_GPDMA1_REQUEST_SPI3_TX
  - LL_GPDMA1_REQUEST_I2C1_RX
  - LL_GPDMA1_REQUEST_I2C1_TX
  - LL_GPDMA1_REQUEST_I2C1_EVC
  - LL_GPDMA1_REQUEST_I2C2_RX
  - LL_GPDMA1_REQUEST_I2C2_TX
  - LL_GPDMA1_REQUEST_I2C2_EVC
  - LL_GPDMA1_REQUEST_I2C3_RX
  - LL_GPDMA1_REQUEST_I2C3_TX
  - LL_GPDMA1_REQUEST_I2C3_EVC
  - LL_GPDMA1_REQUEST_I2C4_RX
  - LL_GPDMA1_REQUEST_I2C4_TX
  - LL_GPDMA1_REQUEST_I2C4_EVC
  - LL_GPDMA1_REQUEST_USART1_RX
  - LL_GPDMA1_REQUEST_USART1_TX
  - LL_GPDMA1_REQUEST_USART2_RX (*)
  - LL_GPDMA1_REQUEST_USART2_TX (*)
  - LL_GPDMA1_REQUEST_USART3_RX
  - LL_GPDMA1_REQUEST_USART3_TX
  - LL_GPDMA1_REQUEST_UART4_RX
  - LL_GPDMA1_REQUEST_UART4_TX
  - LL_GPDMA1_REQUEST_UART5_RX
  - LL_GPDMA1_REQUEST_UART5_TX
  - LL_GPDMA1_REQUEST_LPUART1_RX
  - LL_GPDMA1_REQUEST_LPUART1_TX
  - LL_GPDMA1_REQUEST_SAI1_A
  - LL_GPDMA1_REQUEST_SAI1_B
  - LL_GPDMA1_REQUEST_SAI2_A (*)
  - LL_GPDMA1_REQUEST_SAI2_B (*)
  - LL_GPDMA1_REQUEST_OCTOSPI1
  - LL_GPDMA1_REQUEST_OCTOSPI2 (*)
  - LL_GPDMA1_REQUEST_TIM1_CH1
  - LL_GPDMA1_REQUEST_TIM1_CH2
  - LL_GPDMA1_REQUEST_TIM1_CH3
  - LL_GPDMA1_REQUEST_TIM1_CH4
  - LL_GPDMA1_REQUEST_TIM1_UP
  - LL_GPDMA1_REQUEST_TIM1_TRIG
  - LL_GPDMA1_REQUEST_TIM1_COM

- LL_GPDMA1_REQUEST_TIM8_CH1
- LL_GPDMA1_REQUEST_TIM8_CH2
- LL_GPDMA1_REQUEST_TIM8_CH3
- LL_GPDMA1_REQUEST_TIM8_CH4
- LL_GPDMA1_REQUEST_TIM8_UP
- LL_GPDMA1_REQUEST_TIM8_TRIG
- LL_GPDMA1_REQUEST_TIM8_COM
- LL_GPDMA1_REQUEST_TIM2_CH1
- LL_GPDMA1_REQUEST_TIM2_CH2
- LL_GPDMA1_REQUEST_TIM2_CH3
- LL_GPDMA1_REQUEST_TIM2_CH4
- LL_GPDMA1_REQUEST_TIM2_UP
- LL_GPDMA1_REQUEST_TIM3_CH1
- LL_GPDMA1_REQUEST_TIM3_CH2
- LL_GPDMA1_REQUEST_TIM3_CH3
- LL_GPDMA1_REQUEST_TIM3_CH4
- LL_GPDMA1_REQUEST_TIM3_UP
- LL_GPDMA1_REQUEST_TIM3_TRIG
- LL_GPDMA1_REQUEST_TIM4_CH1
- LL_GPDMA1_REQUEST_TIM4_CH2
- LL_GPDMA1_REQUEST_TIM4_CH3
- LL_GPDMA1_REQUEST_TIM4_CH4
- LL_GPDMA1_REQUEST_TIM4_UP
- LL_GPDMA1_REQUEST_TIM5_CH1
- LL_GPDMA1_REQUEST_TIM5_CH2
- LL_GPDMA1_REQUEST_TIM5_CH3
- LL_GPDMA1_REQUEST_TIM5_CH4
- LL_GPDMA1_REQUEST_TIM5_UP
- LL_GPDMA1_REQUEST_TIM5_TRIG
- LL_GPDMA1_REQUEST_TIM15_CH1
- LL_GPDMA1_REQUEST_TIM15_UP
- LL_GPDMA1_REQUEST_TIM15_TRIG
- LL_GPDMA1_REQUEST_TIM15_COM
- LL_GPDMA1_REQUEST_TIM16_CH1
- LL_GPDMA1_REQUEST_TIM16_UP
- LL_GPDMA1_REQUEST_TIM17_CH1
- LL_GPDMA1_REQUEST_TIM17_UP
- LL_GPDMA1_REQUEST_DCMI_PSSI
- LL_GPDMA1_REQUEST_AES_IN
- LL_GPDMA1_REQUEST_AES_OUT
- LL_GPDMA1_REQUEST_HASH_IN
- LL_GPDMA1_REQUEST_UCPD1_TX (*)
- LL_GPDMA1_REQUEST_UCPD1_RX (*)
- LL_GPDMA1_REQUEST_MDF1_FLT0
- LL_GPDMA1_REQUEST_MDF1_FLT1
- LL_GPDMA1_REQUEST_MDF1_FLT2
- LL_GPDMA1_REQUEST_MDF1_FLT3
- LL_GPDMA1_REQUEST_MDF1_FLT4
- LL_GPDMA1_REQUEST_MDF1_FLT5
- LL_GPDMA1_REQUEST_ADF1_FLT0

- LL_GPDMA1_REQUEST_FMAC_READ
- LL_GPDMA1_REQUEST_FMAC_WRITE
- LL_GPDMA1_REQUEST_CORDIC_READ
- LL_GPDMA1_REQUEST_CORDIC_WRITE
- LL_GPDMA1_REQUEST_SAES_IN
- LL_GPDMA1_REQUEST_SAES_OUT
- LL_GPDMA1_REQUEST_LPTIM1_IC1
- LL_GPDMA1_REQUEST_LPTIM1_IC2
- LL_GPDMA1_REQUEST_LPTIM1_UE
- LL_GPDMA1_REQUEST_LPTIM2_IC1
- LL_GPDMA1_REQUEST_LPTIM2_IC2
- LL_GPDMA1_REQUEST_LPTIM2_UE
- LL_GPDMA1_REQUEST_LPTIM3_IC1
- LL_GPDMA1_REQUEST_LPTIM3_IC2
- LL_GPDMA1_REQUEST_LPTIM3_UE
- LL_GPDMA1_REQUEST_HSPI1 (*)
- LL_GPDMA1_REQUEST_I2C5_RX (*)
- LL_GPDMA1_REQUEST_I2C5_TX (*)
- LL_GPDMA1_REQUEST_I2C5_EVC (*)
- LL_GPDMA1_REQUEST_I2C6_RX (*)
- LL_GPDMA1_REQUEST_I2C6_TX (*)
- LL_GPDMA1_REQUEST_I2C6_EVC (*)
- LL_GPDMA1_REQUEST_USART6_RX (*)
- LL_GPDMA1_REQUEST_USART6_TX (*)
- LL_GPDMA1_REQUEST_ADC2 (*)
- LL_GPDMA1_REQUEST_JPEG_RX (*)
- LL_GPDMA1_REQUEST_JPEG_TX (*)
- LL_LPDMA1_REQUEST_LPUART1_RX
- LL_LPDMA1_REQUEST_LPUART1_TX
- LL_LPDMA1_REQUEST_SPI3_RX
- LL_LPDMA1_REQUEST_SPI3_TX
- LL_LPDMA1_REQUEST_I2C3_RX
- LL_LPDMA1_REQUEST_I2C3_TX
- LL_LPDMA1_REQUEST_I2C3_EVC
- LL_LPDMA1_REQUEST_ADC4
- LL_LPDMA1_REQUEST_DAC1_CH1
- LL_LPDMA1_REQUEST_DAC1_CH2
- LL_LPDMA1_REQUEST_ADF1_FLT0
- LL_LPDMA1_REQUEST_LPTIM1_IC1
- LL_LPDMA1_REQUEST_LPTIM1_IC2
- LL_LPDMA1_REQUEST_LPTIM1_UE
- LL_LPDMA1_REQUEST_LPTIM3_IC1
- LL_LPDMA1_REQUEST_LPTIM3_IC2
- LL_LPDMA1_REQUEST_LPTIM3_UE

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For LPDMA channels, REQSEL fields is reduced to 5 bits.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 REQSEL LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get hardware request.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPDMA1_REQUEST_ADC1
  - LL_GPDMA1_REQUEST_ADC4
  - LL_GPDMA1_REQUEST_DAC1_CH1
  - LL_GPDMA1_REQUEST_DAC1_CH2
  - LL_GPDMA1_REQUEST_TIM6_UP
  - LL_GPDMA1_REQUEST_TIM7_UP
  - LL_GPDMA1_REQUEST_SPI1_RX
  - LL_GPDMA1_REQUEST_SPI1_TX
  - LL_GPDMA1_REQUEST_SPI2_RX
  - LL_GPDMA1_REQUEST_SPI2_TX
  - LL_GPDMA1_REQUEST_SPI3_RX
  - LL_GPDMA1_REQUEST_SPI3_TX
  - LL_GPDMA1_REQUEST_I2C1_RX
  - LL_GPDMA1_REQUEST_I2C1_TX
  - LL_GPDMA1_REQUEST_I2C1_EVC
  - LL_GPDMA1_REQUEST_I2C2_RX
  - LL_GPDMA1_REQUEST_I2C2_TX
  - LL_GPDMA1_REQUEST_I2C2_EVC
  - LL_GPDMA1_REQUEST_I2C3_RX
  - LL_GPDMA1_REQUEST_I2C3_TX
  - LL_GPDMA1_REQUEST_I2C3_EVC
  - LL_GPDMA1_REQUEST_I2C4_RX
  - LL_GPDMA1_REQUEST_I2C4_TX
  - LL_GPDMA1_REQUEST_I2C4_EVC
  - LL_GPDMA1_REQUEST_USART1_RX
  - LL_GPDMA1_REQUEST_USART1_TX
  - LL_GPDMA1_REQUEST_USART2_RX (*)
  - LL_GPDMA1_REQUEST_USART2_TX (*)
  - LL_GPDMA1_REQUEST_USART3_RX
  - LL_GPDMA1_REQUEST_USART3_TX
  - LL_GPDMA1_REQUEST_UART4_RX
  - LL_GPDMA1_REQUEST_UART4_TX
  - LL_GPDMA1_REQUEST_UART5_RX
  - LL_GPDMA1_REQUEST_UART5_TX
  - LL_GPDMA1_REQUEST_LPUART1_RX
  - LL_GPDMA1_REQUEST_LPUART1_TX
  - LL_GPDMA1_REQUEST_SAI1_A
  - LL_GPDMA1_REQUEST_SAI1_B
  - LL_GPDMA1_REQUEST_SAI2_A (*)
  - LL_GPDMA1_REQUEST_SAI2_B (*)
  - LL_GPDMA1_REQUEST_OCTOSPI1
  - LL_GPDMA1_REQUEST_OCTOSPI2 (*)
  - LL_GPDMA1_REQUEST_TIM1_CH1
  - LL_GPDMA1_REQUEST_TIM1_CH2
  - LL_GPDMA1_REQUEST_TIM1_CH3
  - LL_GPDMA1_REQUEST_TIM1_CH4
  - LL_GPDMA1_REQUEST_TIM1_UP
  - LL_GPDMA1_REQUEST_TIM1_TRIG

- LL_GPDMA1_REQUEST_TIM1_COM
- LL_GPDMA1_REQUEST_TIM8_CH1
- LL_GPDMA1_REQUEST_TIM8_CH2
- LL_GPDMA1_REQUEST_TIM8_CH3
- LL_GPDMA1_REQUEST_TIM8_CH4
- LL_GPDMA1_REQUEST_TIM8_UP
- LL_GPDMA1_REQUEST_TIM8_TRIG
- LL_GPDMA1_REQUEST_TIM8_COM
- LL_GPDMA1_REQUEST_TIM2_CH1
- LL_GPDMA1_REQUEST_TIM2_CH2
- LL_GPDMA1_REQUEST_TIM2_CH3
- LL_GPDMA1_REQUEST_TIM2_CH4
- LL_GPDMA1_REQUEST_TIM2_UP
- LL_GPDMA1_REQUEST_TIM3_CH1
- LL_GPDMA1_REQUEST_TIM3_CH2
- LL_GPDMA1_REQUEST_TIM3_CH3
- LL_GPDMA1_REQUEST_TIM3_CH4
- LL_GPDMA1_REQUEST_TIM3_UP
- LL_GPDMA1_REQUEST_TIM3_TRIG
- LL_GPDMA1_REQUEST_TIM4_CH1
- LL_GPDMA1_REQUEST_TIM4_CH2
- LL_GPDMA1_REQUEST_TIM4_CH3
- LL_GPDMA1_REQUEST_TIM4_CH4
- LL_GPDMA1_REQUEST_TIM4_UP
- LL_GPDMA1_REQUEST_TIM5_CH1
- LL_GPDMA1_REQUEST_TIM5_CH2
- LL_GPDMA1_REQUEST_TIM5_CH3
- LL_GPDMA1_REQUEST_TIM5_CH4
- LL_GPDMA1_REQUEST_TIM5_UP
- LL_GPDMA1_REQUEST_TIM5_TRIG
- LL_GPDMA1_REQUEST_TIM15_CH1
- LL_GPDMA1_REQUEST_TIM15_UP
- LL_GPDMA1_REQUEST_TIM15_TRIG
- LL_GPDMA1_REQUEST_TIM15_COM
- LL_GPDMA1_REQUEST_TIM16_CH1
- LL_GPDMA1_REQUEST_TIM16_UP
- LL_GPDMA1_REQUEST_TIM17_CH1
- LL_GPDMA1_REQUEST_TIM17_UP
- LL_GPDMA1_REQUEST_DCMI_PSSI
- LL_GPDMA1_REQUEST_AES_IN
- LL_GPDMA1_REQUEST_AES_OUT
- LL_GPDMA1_REQUEST_HASH_IN
- LL_GPDMA1_REQUEST_UCPD1_TX (*)
- LL_GPDMA1_REQUEST_UCPD1_RX (*)
- LL_GPDMA1_REQUEST_MDF1_FLT0
- LL_GPDMA1_REQUEST_MDF1_FLT1
- LL_GPDMA1_REQUEST_MDF1_FLT2
- LL_GPDMA1_REQUEST_MDF1_FLT3
- LL_GPDMA1_REQUEST_MDF1_FLT4
- LL_GPDMA1_REQUEST_MDF1_FLT5

- LL_GPDMA1_REQUEST_ADF1_FLT0
- LL_GPDMA1_REQUEST_FMAC_READ
- LL_GPDMA1_REQUEST_FMAC_WRITE
- LL_GPDMA1_REQUEST_CORDIC_READ
- LL_GPDMA1_REQUEST_CORDIC_WRITE
- LL_GPDMA1_REQUEST_SAES_IN
- LL_GPDMA1_REQUEST_SAES_OUT
- LL_GPDMA1_REQUEST_LPTIM1_IC1
- LL_GPDMA1_REQUEST_LPTIM1_IC2
- LL_GPDMA1_REQUEST_LPTIM1_UE
- LL_GPDMA1_REQUEST_LPTIM2_IC1
- LL_GPDMA1_REQUEST_LPTIM2_IC2
- LL_GPDMA1_REQUEST_LPTIM2_UE
- LL_GPDMA1_REQUEST_LPTIM3_IC1
- LL_GPDMA1_REQUEST_LPTIM3_IC2
- LL_GPDMA1_REQUEST_LPTIM3_UE
- LL_GPDMA1_REQUEST_HSPI1 (*)
- LL_GPDMA1_REQUEST_I2C5_RX (*)
- LL_GPDMA1_REQUEST_I2C5_TX (*)
- LL_GPDMA1_REQUEST_I2C5_EVC (*)
- LL_GPDMA1_REQUEST_I2C6_RX (*)
- LL_GPDMA1_REQUEST_I2C6_TX (*)
- LL_GPDMA1_REQUEST_I2C6_EVC (*)
- LL_GPDMA1_REQUEST_USART6_RX (*)
- LL_GPDMA1_REQUEST_USART6_TX (*)
- LL_GPDMA1_REQUEST_ADC2 (*)
- LL_GPDMA1_REQUEST_JPEG_RX (*)
- LL_GPDMA1_REQUEST_JPEG_TX (*)
- LL_LPDMA1_REQUEST_LPUART1_RX
- LL_LPDMA1_REQUEST_LPUART1_TX
- LL_LPDMA1_REQUEST_SPI3_RX
- LL_LPDMA1_REQUEST_SPI3_TX
- LL_LPDMA1_REQUEST_I2C3_RX
- LL_LPDMA1_REQUEST_I2C3_TX
- LL_LPDMA1_REQUEST_I2C3_EVC
- LL_LPDMA1_REQUEST_ADC4
- LL_LPDMA1_REQUEST_DAC1_CH1
- LL_LPDMA1_REQUEST_DAC1_CH2
- LL_LPDMA1_REQUEST_ADF1_FLT0
- LL_LPDMA1_REQUEST_LPTIM1_IC1
- LL_LPDMA1_REQUEST_LPTIM1_IC2
- LL_LPDMA1_REQUEST_LPTIM1_UE
- LL_LPDMA1_REQUEST_LPTIM3_IC1
- LL_LPDMA1_REQUEST_LPTIM3_IC2
- LL_LPDMA1_REQUEST_LPTIM3_UE

Notes

- This API is used for all available DMA channels. For LPDMA channels, REQSEL fields is reduced to 5 bits.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 REQSEL LL_DMA_GetPeriphRequest

LL_DMA_SetHWTrigger

Function name

__STATIC_INLINE void LL_DMA_SetHWTrigger (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Trigger)

Function description

Set hardware trigger.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

- **Trigger:** This parameter can be one of the following values:
  - LL_GPDMA1_TRIGGER_EXTI_LINE0
  - LL_GPDMA1_TRIGGER_EXTI_LINE1
  - LL_GPDMA1_TRIGGER_EXTI_LINE2
  - LL_GPDMA1_TRIGGER_EXTI_LINE3
  - LL_GPDMA1_TRIGGER_TAMP_TRG1
  - LL_GPDMA1_TRIGGER_TAMP_TRG2
  - LL_GPDMA1_TRIGGER_TAMP_TRG3
  - LL_GPDMA1_TRIGGER_LPTIM1_CH1
  - LL_GPDMA1_TRIGGER_LPTIM1_CH2
  - LL_GPDMA1_TRIGGER_LPTIM2_CH1
  - LL_GPDMA1_TRIGGER_LPTIM2_CH2
  - LL_GPDMA1_TRIGGER_LPTIM4_OUT
  - LL_GPDMA1_TRIGGER_COMP1_OUT
  - LL_GPDMA1_TRIGGER_COMP2_OUT (*)
  - LL_GPDMA1_TRIGGER_RTC_ALRA_TRG
  - LL_GPDMA1_TRIGGER_RTC_ALRB_TRG
  - LL_GPDMA1_TRIGGER_RTC_WUT_TRG
  - LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH8_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH9_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH10_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH11_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH12_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH13_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH14_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH15_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH0_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH1_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH2_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH3_TCF
  - LL_GPDMA1_TRIGGER_TIM2_TRGO
  - LL_GPDMA1_TRIGGER_TIM15_TRGO
  - LL_GPDMA1_TRIGGER_TIM3_TRGO (*)
  - LL_GPDMA1_TRIGGER_TIM4_TRGO (*)
  - LL_GPDMA1_TRIGGER_TIM5_TRGO (*)
  - LL_GPDMA1_TRIGGER_LTDC_LI (*)
  - LL_GPDMA1_TRIGGER_DSI_TE (*)
  - LL_GPDMA1_TRIGGER_DSI_ER (*)
  - LL_GPDMA1_TRIGGER_DMA2D_TC (*)
  - LL_GPDMA1_TRIGGER_DMA2D_CTC (*)
  - LL_GPDMA1_TRIGGER_DMA2D_TW (*)
  - LL_GPDMA1_TRIGGER_GPU2D_FLAG0 (*)

- LL_GPDMA1_TRIGGER_GPU2D_FLAG1 (*)
- LL_GPDMA1_TRIGGER_GPU2D_FLAG2 (*)
- LL_GPDMA1_TRIGGER_GPU2D_FLAG3 (*)
- LL_GPDMA1_TRIGGER_ADC4_AWD1
- LL_GPDMA1_TRIGGER_ADC1_AWD1
- LL_GPDMA1_TRIGGER_GFXTIM_EVT3 (*)
- LL_GPDMA1_TRIGGER_GFXTIM_EVT2 (*)
- LL_GPDMA1_TRIGGER_GFXTIM_EVT1 (*)
- LL_GPDMA1_TRIGGER_GFXTIM_EVT0 (*)
- LL_GPDMA1_TRIGGER_JPEG_EOC (*)
- LL_GPDMA1_TRIGGER_JPEG_IFNF (*)
- LL_GPDMA1_TRIGGER_JPEG_IFT (*)
- LL_GPDMA1_TRIGGER_JPEG_OFNE (*)
- LL_GPDMA1_TRIGGER_JPEG_OFT (*)
- LL_LPDMA1_TRIGGER_EXTI_LINE0
- LL_LPDMA1_TRIGGER_EXTI_LINE1
- LL_LPDMA1_TRIGGER_EXTI_LINE2
- LL_LPDMA1_TRIGGER_EXTI_LINE3
- LL_LPDMA1_TRIGGER_EXTI_LINE4
- LL_LPDMA1_TRIGGER_TAMP_TRG1
- LL_LPDMA1_TRIGGER_TAMP_TRG2
- LL_LPDMA1_TRIGGER_TAMP_TRG3
- LL_LPDMA1_TRIGGER_LPTIM1_CH1
- LL_LPDMA1_TRIGGER_LPTIM1_CH2
- LL_LPDMA1_TRIGGER_LPTIM3_CH1
- LL_LPDMA1_TRIGGER_LPTIM4_OUT
- LL_LPDMA1_TRIGGER_COMP1_OUT
- LL_LPDMA1_TRIGGER_COMP2_OUT (*)
- LL_LPDMA1_TRIGGER_RTC_ALRA_TRG
- LL_LPDMA1_TRIGGER_RTC_ALRB_TRG
- LL_LPDMA1_TRIGGER_RTC_WUT_TRG
- LL_LPDMA1_TRIGGER_ADC4_AWD1
- LL_LPDMA1_TRIGGER_LPDMA1_CH0_TCF
- LL_LPDMA1_TRIGGER_LPDMA1_CH1_TCF
- LL_LPDMA1_TRIGGER_LPDMA1_CH2_TCF
- LL_LPDMA1_TRIGGER_LPDMA1_CH3_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH0_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH1_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH4_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH5_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH6_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH7_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH12_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH13_TCF
- LL_LPDMA1_TRIGGER_TIM2_TRGO
- LL_LPDMA1_TRIGGER_TIM15_TRGO

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For LPDMA channels, TRIGSEL fields is reduced to 5 bits.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_SetHWTrigger

LL_DMA_GetHWTrigger

Function name

__STATIC_INLINE uint32_t LL_DMA_GetHWTrigger (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get hardware triggers.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPDMA1_TRIGGER_EXTI_LINE0
  - LL_GPDMA1_TRIGGER_EXTI_LINE1
  - LL_GPDMA1_TRIGGER_EXTI_LINE2
  - LL_GPDMA1_TRIGGER_EXTI_LINE3
  - LL_GPDMA1_TRIGGER_TAMP_TRG1
  - LL_GPDMA1_TRIGGER_TAMP_TRG2
  - LL_GPDMA1_TRIGGER_TAMP_TRG3
  - LL_GPDMA1_TRIGGER_LPTIM1_CH1
  - LL_GPDMA1_TRIGGER_LPTIM1_CH2
  - LL_GPDMA1_TRIGGER_LPTIM2_CH1
  - LL_GPDMA1_TRIGGER_LPTIM2_CH2
  - LL_GPDMA1_TRIGGER_LPTIM4_OUT
  - LL_GPDMA1_TRIGGER_COMP1_OUT
  - LL_GPDMA1_TRIGGER_COMP2_OUT (*)
  - LL_GPDMA1_TRIGGER_RTC_ALRA_TRG
  - LL_GPDMA1_TRIGGER_RTC_ALRB_TRG
  - LL_GPDMA1_TRIGGER_RTC_WUT_TRG
  - LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH8_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH9_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH10_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH11_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH12_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH13_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH14_TCF
  - LL_GPDMA1_TRIGGER_GPDMA1_CH15_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH0_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH1_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH2_TCF
  - LL_GPDMA1_TRIGGER_LPDMA1_CH3_TCF
  - LL_GPDMA1_TRIGGER_TIM2_TRGO
  - LL_GPDMA1_TRIGGER_TIM15_TRGO
  - LL_GPDMA1_TRIGGER_TIM3_TRGO (*)
  - LL_GPDMA1_TRIGGER_TIM4_TRGO (*)
  - LL_GPDMA1_TRIGGER_TIM5_TRGO (*)
  - LL_GPDMA1_TRIGGER_LTDC_LI (*)
  - LL_GPDMA1_TRIGGER_DSI_TE (*)
  - LL_GPDMA1_TRIGGER_DSI_ER (*)
  - LL_GPDMA1_TRIGGER_DMA2D_TC (*)
  - LL_GPDMA1_TRIGGER_DMA2D_CTC (*)
  - LL_GPDMA1_TRIGGER_DMA2D_TW (*)

- LL_GPDMA1_TRIGGER_GPU2D_FLAG0 (*)
- LL_GPDMA1_TRIGGER_GPU2D_FLAG1 (*)
- LL_GPDMA1_TRIGGER_GPU2D_FLAG2 (*)
- LL_GPDMA1_TRIGGER_GPU2D_FLAG3 (*)
- LL_GPDMA1_TRIGGER_ADC4_AWD1
- LL_GPDMA1_TRIGGER_ADC1_AWD1
- LL_GPDMA1_TRIGGER_GFXTIM_EVT3 (*)
- LL_GPDMA1_TRIGGER_GFXTIM_EVT2 (*)
- LL_GPDMA1_TRIGGER_GFXTIM_EVT1 (*)
- LL_GPDMA1_TRIGGER_GFXTIM_EVT0 (*)
- LL_GPDMA1_TRIGGER_JPEG_EOC (*)
- LL_GPDMA1_TRIGGER_JPEG_IFNF (*)
- LL_GPDMA1_TRIGGER_JPEG_IFT (*)
- LL_GPDMA1_TRIGGER_JPEG_OFNE (*)
- LL_GPDMA1_TRIGGER_JPEG_OFT (*)
- LL_LPDMA1_TRIGGER_EXTI_LINE0
- LL_LPDMA1_TRIGGER_EXTI_LINE1
- LL_LPDMA1_TRIGGER_EXTI_LINE2
- LL_LPDMA1_TRIGGER_EXTI_LINE3
- LL_LPDMA1_TRIGGER_EXTI_LINE4
- LL_LPDMA1_TRIGGER_TAMP_TRG1
- LL_LPDMA1_TRIGGER_TAMP_TRG2
- LL_LPDMA1_TRIGGER_TAMP_TRG3
- LL_LPDMA1_TRIGGER_LPTIM1_CH1
- LL_LPDMA1_TRIGGER_LPTIM1_CH2
- LL_LPDMA1_TRIGGER_LPTIM3_CH1
- LL_LPDMA1_TRIGGER_LPTIM4_OUT
- LL_LPDMA1_TRIGGER_COMP1_OUT
- LL_LPDMA1_TRIGGER_COMP2_OUT (*)
- LL_LPDMA1_TRIGGER_RTC_ALRA_TRG
- LL_LPDMA1_TRIGGER_RTC_ALRB_TRG
- LL_LPDMA1_TRIGGER_RTC_WUT_TRG
- LL_LPDMA1_TRIGGER_ADC4_AWD1
- LL_LPDMA1_TRIGGER_LPDMA1_CH0_TCF
- LL_LPDMA1_TRIGGER_LPDMA1_CH1_TCF
- LL_LPDMA1_TRIGGER_LPDMA1_CH2_TCF
- LL_LPDMA1_TRIGGER_LPDMA1_CH3_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH0_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH1_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH4_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH5_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH6_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH7_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH12_TCF
- LL_LPDMA1_TRIGGER_GPDMA1_CH13_TCF
- LL_LPDMA1_TRIGGER_TIM2_TRGO
- LL_LPDMA1_TRIGGER_TIM15_TRGO

Notes

- This API is used for all available DMA channels. For LPDMA channels, TRIGSEL fields is reduced to 5 bits.
- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CTR2 TRIGSEL LL_DMA_GetHWTrigger

LL_DMA_ConfigBlkRptAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkRptAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure addresses update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT or LL_DMA_BLKRPT_DEST_ADDR_DECREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT or LL_DMA_BLKRPT_SRC_ADDR_DECREMENT
  - LL_DMA_BURST_DEST_ADDR_INCREMENT or LL_DMA_BURST_DEST_ADDR_DECREMENT
  - LL_DMA_BURST_SRC_ADDR_INCREMENT or LL_DMA_BURST_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 BRSDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 DDEC LL_DMA_ConfigBlkRptAddrUpdate
- CBR1 SDEC LL_DMA_ConfigBlkRptAddrUpdate

LL_DMA_ConfigBlkCounters

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkCounters (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkDataLength, uint32_t BlkRptCount)

Function description

Configure DMA Block number of data and repeat Count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkDataLength:** Block transfer length Value between 0 to 0x0000FFFF
- **BlkRptCount:** Block repeat counter Value between 0 to 0x000007FF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_ConfigBlkCounters
- CBR1 BRC LL_DMA_ConfigBlkCounters

LL_DMA_SetBlkRptDestAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptDestAddrUpdate)

Function description

Set block repeat destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptDestAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT
  - LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_SetBlkRptDestAddrUpdate

LL_DMA_GetBlkRptDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BLKRPT_DEST_ADDR_INCREMENT
  - LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRDDEC LL_DMA_GetBlkRptDestAddrUpdate

LL_DMA_SetBlkRptSrcAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrUpdate)

Function description

Set block repeat source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkRptSrcAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRSDEC LL_DMA_SetBlkRptSrcAddrUpdate

LL_DMA_GetBlkRptSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_BLKRPT_SRC_ADDR_INCREMENT
  - LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRSDEC LL_DMA_GetBlkRptSrcAddrUpdate

LL_DMA_SetDestAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddrUpdate)

Function description

Set destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAddrUpdate:** This parameter can be one of the following values:
  - LL_DMA_BURST_DEST_ADDR_INCREMENT
  - LL_DMA_BURST_DEST_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 DDEC LL_DMA_SetDestAddrUpdate

LL_DMA_GetDestAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_BURST_DEST_ADDR_INCREMENT
  – LL_DMA_BURST_DEST_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 DDEC LL_DMA_GetDestAddrUpdate

LL_DMA_SetSrcAddrUpdate

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrUpdate)

Function description

Set source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **SrcAddrUpdate:** This parameter can be one of the following values:
  – LL_DMA_BURST_SRC_ADDR_INCREMENT
  – LL_DMA_BURST_SRC_ADDR_DECREMENT

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 SDEC LL_DMA_SetSrcAddrUpdate

LL_DMA_GetSrcAddrUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddrUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address update.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_BURST_SRC_ADDR_INCREMENT
  – LL_DMA_BURST_SRC_ADDR_DECREMENT

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 SDEC LL_DMA_GetSrcAddrUpdate

LL_DMA_SetBlkRptCount

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptCount (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptCount)

Function description

Set block repeat count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **BlkRptCount:** Block repeat counter Value between 0 to 0x000007FF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRC LL_DMA_SetBlkRptCount

LL_DMA_GetBlkRptCount

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptCount (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeat count.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x000007FF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR1 BRC LL_DMA_GetBlkRptCount

LL_DMA_SetBlkDataLength

Function name

__STATIC_INLINE void LL_DMA_SetBlkDataLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkDataLength)

Function description

Set block data length in bytes to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **BlkDataLength:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_SetBlkDataLength

LL_DMA_GetBlkDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkDataLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block data length in bytes to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CBR1 BNDT LL_DMA_GetBlkDataLength

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DestAddress)

Function description

Configure the source and destination addresses.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddress:** Between 0 to 0xFFFFFFFF
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_ConfigAddresses
- CDAR DA LL_DMA_ConfigAddresses

LL_DMA_SetSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress)

Function description

Set source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_SetSrcAddress

LL_DMA_GetSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSAR SA LL_DMA_GetSrcAddress

LL_DMA_SetDestAddress

Function name

__STATIC_INLINE void LL_DMA_SetDestAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddress)

Function description

Set destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DestAddress:** Between 0 to 0xFFFFFFFF

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CDAR DA LL_DMA_SetDestAddress

LL_DMA_GetDestAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0xFFFFFFFF

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CDAR DA LL_DMA_GetDestAddress

LL_DMA_ConfigAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_ConfigAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrOffset, uint32_t DestAddrOffset)

Function description

Configure source and destination addresses offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **DestAddrOffset:** Between 0 to 0x00001FFF
- **SrcAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CTR3 DAO LL_DMA_ConfigAddrUpdateValue
- CTR3 SAO LL_DMA_ConfigAddrUpdateValue

LL_DMA_SetDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t DestAddrOffset)

Function description

Set destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **DestAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 DAO LL_DMA_SetDestAddrUpdateValue

LL_DMA_GetDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x00001FFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CDAR DAO LL_DMA_GetDestAddrUpdateValue

LL_DMA_SetSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddrOffset)

Function description

Set source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **SrcAddrOffset:** Between 0 to 0x00001FFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 SAO LL_DMA_SetSrcAddrUpdateValue

LL_DMA_GetSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x00001FFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CTR3 SAO LL_DMA_GetSrcAddrUpdateValue

LL_DMA_ConfigBlkRptAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_ConfigBlkRptAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrOffset, uint32_t BlkRptDestAddrOffset)

Function description

Configure the block repeated source and destination addresses offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **BlkRptDestAddrOffset:** Between 0 to 0x0000FFFF
- **BlkRptSrcAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.
- This API must not be called when the DMA Channel is enabled.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_ConfigBlkRptAddrUpdateValue
- CBR2 BRSAO LL_DMA_ConfigBlkRptAddrUpdateValue

LL_DMA_SetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptDestAddrOffset)

Function description

Set block repeated destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **BlkRptDestAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_SetBlkRptDestAddrUpdateValue

LL_DMA_GetBlkRptDestAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptDestAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeated destination address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF.

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRDAO LL_DMA_GetBlkRptDestAddrUpdateValue

LL_DMA_SetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE void LL_DMA_SetBlkRptSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t BlkRptSrcAddrOffset)

Function description

Set block repeated source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **BlkRptSrcAddrOffset:** Between 0 to 0x0000FFFF

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRSAO LL_DMA_SetBlkRptSrcAddrUpdateValue

LL_DMA_GetBlkRptSrcAddrUpdateValue

Function name

__STATIC_INLINE uint32_t LL_DMA_GetBlkRptSrcAddrUpdateValue (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get block repeated source address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFF

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CBR2 BRSAO LL_DMA_GetBlkRptSrcAddrUpdateValue

LL_DMA_ConfigLinkUpdate

Function name

__STATIC_INLINE void LL_DMA_ConfigLinkUpdate (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t RegistersUpdate, uint32_t LinkedListAddrOffset)

Function description

Configure registers update and node address offset during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **RegistersUpdate:** This parameter must be a combination of all the following values:
  - LL_DMA_UPDATE_CTR1
  - LL_DMA_UPDATE_CTR2
  - LL_DMA_UPDATE_CBR1
  - LL_DMA_UPDATE_CSAR
  - LL_DMA_UPDATE_CDAR
  - LL_DMA_UPDATE_CTR3 (This value is allowed only for 2D addressing channels)
  - LL_DMA_UPDATE_CBR2 (This value is allowed only for 2D addressing channels)
  - LL_DMA_UPDATE_CLLR
- **LinkedListAddrOffset:** Between 0 to 0x0000FFFC by increment of 4 Bytes.

Return values

- **None.:**

Notes

- This API is used for all available DMA channels. For linear addressing channels, UT3 and UB2 fields are discarded.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_ConfigLinkUpdate
-
- CLLR UT2 LL_DMA_ConfigLinkUpdate
-
- CLLR UB1 LL_DMA_ConfigLinkUpdate
-
- CLLR USA LL_DMA_ConfigLinkUpdate
-
- CLLR UDA LL_DMA_ConfigLinkUpdate
-
- CLLR UT3 LL_DMA_ConfigLinkUpdate
-
- CLLR UB2 LL_DMA_ConfigLinkUpdate
-
- CLLR ULL LL_DMA_ConfigLinkUpdate

LL_DMA_EnableCTR1Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_EnableCTR1Update

LL_DMA_DisableCTR1Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_DisableCTR1Update

LL_DMA_IsEnabledCTR1Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR1 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT1 LL_DMA_IsEnabledCTR1Update

LL_DMA_EnableCTR2Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_EnableCTR2Update

LL_DMA_DisableCTR2Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_DisableCTR2Update

LL_DMA_IsEnabledCTR2Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR2 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UT2 LL_DMA_IsEnabledCTR2Update

LL_DMA_EnableCBR1Update

Function name

__STATIC_INLINE void LL_DMA_EnableCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CBR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_EnableCBR1Update

LL_DMA_DisableCBR1Update

Function name

__STATIC_INLINE void LL_DMA_DisableCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CBR1 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_DisableCBR1Update

LL_DMA_IsEnabledCBR1Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCBR1Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CBR1 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UB1 LL_DMA_IsEnabledCBR1Update

LL_DMA_EnableCSARUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CSAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_EnableCSARUpdate

LL_DMA_DisableCSARUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CSAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_DisableCSARUpdate

LL_DMA_IsEnabledCSARUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCSARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CSAR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR USA LL_DMA_IsEnabledCSARUpdate

LL_DMA_EnableCDARUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CDAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_EnableCDARUpdate

LL_DMA_DisableCDARUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CDAR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_DisableCDARUpdate

LL_DMA_IsEnabledCDARUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCDARUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CDAR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR UDA LL_DMA_IsEnabledCDARUpdate

LL_DMA_EnableCTR3Update

Function name

__STATIC_INLINE void LL_DMA_EnableCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CTR3 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_EnableCTR3Update

LL_DMA_DisableCTR3Update

Function name

__STATIC_INLINE void LL_DMA_DisableCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CTR3 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_DisableCTR3Update

LL_DMA_IsEnabledCTR3Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCTR3Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CTR3 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UT3 LL_DMA_IsEnabledCTR3Update

LL_DMA_EnableCBR2Update

Function name

__STATIC_INLINE void LL_DMA_EnableCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CBR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_EnableCBR2Update

LL_DMA_DisableCBR2Update

Function name

__STATIC_INLINE void LL_DMA_DisableCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CBR2 update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_DisableCBR2Update

LL_DMA_IsEnabledCBR2Update

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCBR2Update (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CBR2 update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used only for 2D addressing channels.

Reference Manual to LL API cross reference:

- CLLR UB2 LL_DMA_IsEnabledCBR2Update

LL_DMA_EnableCLLRUpdate

Function name

__STATIC_INLINE void LL_DMA_EnableCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable CLLR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_EnableCLLRUpdate

LL_DMA_DisableCLLRUpdate

Function name

__STATIC_INLINE void LL_DMA_DisableCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable CLLR update during the link transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_DisableCLLRUpdate

LL_DMA_IsEnabledCLLRUpdate

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledCLLRUpdate (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if CLLR update during the link transfer is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR ULL LL_DMA_IsEnabledCLLRUpdate

LL_DMA_SetLinkedListAddrOffset

Function name

__STATIC_INLINE void LL_DMA_SetLinkedListAddrOffset (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t LinkedListAddrOffset)

Function description

Set linked list address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **LinkedListAddrOffset:** Between 0 to 0x0000FFFC by increment of 4 Bytes.

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR LA LL_DMA_SetLinkedListAddrOffset

LL_DMA_GetLinkedListAddrOffset

Function name

__STATIC_INLINE uint32_t LL_DMA_GetLinkedListAddrOffset (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get linked list address offset.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x0000FFFC.

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CLLR LA LL_DMA_GetLinkedListAddrOffset

LL_DMA_GetFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetFIFOLevel (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get FIFO level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **Between:** 0 to 0x000000FF.

Notes

- This API is not used for LPDMA channels.

Reference Manual to LL API cross reference:

- CSR FIFOL LL_DMA_GetFIFOLevel

LL_DMA_EnableChannelSecure

Function name

__STATIC_INLINE void LL_DMA_EnableChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel secure attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_EnableChannelSecure

LL_DMA_DisableChannelSecure

Function name

__STATIC_INLINE void LL_DMA_DisableChannelSecure (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the DMA channel secure attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_DisableChannelSecure

LL_DMA_IsEnabledChannelSecure

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelSecure (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel secure is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR SECx LL_DMA_IsEnabledChannelSecure

LL_DMA_EnableChannelPrivilege

Function name

__STATIC_INLINE void LL_DMA_EnableChannelPrivilege (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel privilege attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_EnableChannelPrivilege

LL_DMA_DisableChannelPrivilege

Function name

__STATIC_INLINE void LL_DMA_DisableChannelPrivilege (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable the DMA channel privilege attribute.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_DisableChannelPrivilege

LL_DMA_IsEnabledChannelPrivilege

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelPrivilege (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA Channel privilege is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- PRIVCFGR PRIVx LL_DMA_IsEnabledChannelPrivilege

LL_DMA_EnableChannelLockAttribute

Function name

__STATIC_INLINE void LL_DMA_EnableChannelLockAttribute (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable the DMA channel lock attributes.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- RCFGLOCKR LOCKx LL_DMA_EnableChannelLockAttribute

LL_DMA_IsEnabledChannelLockAttribute

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannelLockAttribute (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if DMA channel attributes are locked.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SECCFGR LOCKx LL_DMA_IsEnabledChannelLockAttribute

LL_DMA_ClearFlag_TO

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear trigger overrun flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR TOF LL_DMA_ClearFlag_TO

LL_DMA_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear suspension flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR SUSPF LL_DMA_ClearFlag_SUSP

LL_DMA_ClearFlag_USE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR USEF LL_DMA_ClearFlag_USE

LL_DMA_ClearFlag_ULE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear link transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR ULEF LL_DMA_ClearFlag_ULE

LL_DMA_ClearFlag_DTE

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear data transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR DTEF LL_DMA_ClearFlag_DTE

LL_DMA_ClearFlag_HT

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear half transfer flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR HTF LL_DMA_ClearFlag_HT

LL_DMA_ClearFlag_TC

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Clear transfer complete flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CFCR TCF LL_DMA_ClearFlag_TC

LL_DMA_IsActiveFlag_TO

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get trigger overrun flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR TOF LL_DMA_IsActiveFlag_TO

LL_DMA_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get suspension flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR SUSPF LL_DMA_IsActiveFlag_SUSP

LL_DMA_IsActiveFlag_USE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR USEF LL_DMA_IsActiveFlag_USE

LL_DMA_IsActiveFlag_ULE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get user setting error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR ULEF LL_DMA_IsActiveFlag_ULE

LL_DMA_IsActiveFlag_DTE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get data transfer error flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR DTEF LL_DMA_IsActiveFlag_DTE

LL_DMA_IsActiveFlag_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get half transfer flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR HTF LL_DMA_IsActiveFlag_HT

LL_DMA_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get transfer complete flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR TCF LL_DMA_IsActiveFlag_TC

LL_DMA_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_IDLE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get idle flag.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CSR IDLEF LL_DMA_IsActiveFlag_IDLE

LL_DMA_IsActiveFlag_MIS

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_MIS (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if nsecure masked interrupt is active.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- MISR MISx LL_DMA_IsActiveFlag_MIS

LL_DMA_IsActiveFlag_SMIS

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_SMIS (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if secure masked interrupt is active.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- SMISR MISx LL_DMA_IsActiveFlag_SMIS

LL_DMA_EnableIT_TO

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable trigger overrun interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_EnableIT_TO

LL_DMA_EnableIT_SUSP

Function name

__STATIC_INLINE void LL_DMA_EnableIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable suspension interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_EnableIT_SUSP

LL_DMA_EnableIT_USE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable user setting error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_EnableIT_USE

LL_DMA_EnableIT_ULE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable update link transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_EnableIT_ULE

LL_DMA_EnableIT_DTE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable data transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_EnableIT_DTE

LL_DMA_EnableIT_HT

Function name

__STATIC_INLINE void LL_DMA_EnableIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable half transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_EnableIT_TC

LL_DMA_DisableIT_TO

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable trigger overrun interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_DisableIT_TO

LL_DMA_DisableIT_SUSP

Function name

__STATIC_INLINE void LL_DMA_DisableIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable suspension interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_DisableIT_SUSP

LL_DMA_DisableIT_USE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable user setting error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_DisableIT_USE

LL_DMA_DisableIT_ULE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable update link transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_DisableIT_ULE

LL_DMA_DisableIT_DTE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable data transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_DisableIT_DTE

LL_DMA_DisableIT_HT

Function name

__STATIC_INLINE void LL_DMA_DisableIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable half transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TC

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable transfer complete interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **None.:**

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_DisableIT_TC

LL_DMA_IsEnabledIT_TO

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TO (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if trigger overrun interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TOIE LL_DMA_IsEnabledIT_TO

LL_DMA_IsEnabledIT_SUSP

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_SUSP (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if suspension interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR SUSPIE LL_DMA_IsEnabledIT_SUSP

LL_DMA_IsEnabledIT_USE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_USE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if user setting error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR USEIE LL_DMA_IsEnabledIT_USE

LL_DMA_IsEnabledIT_ULE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_ULE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if update link transfer error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR ULEIE LL_DMA_IsEnabledIT_ULE

LL_DMA_IsEnabledIT_DTE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_DTE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if data transfer error interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR DTEIE LL_DMA_IsEnabledIT_DTE

LL_DMA_IsEnabledIT_HT

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_HT (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if half transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TC (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if transfer complete interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15

Return values

- **State:** of bit (1 or 0).

Notes

- This API is used for all available DMA channels.

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_IsEnabledIT_TC

LL_DMA_Init

Function name

uint32_t LL_DMA_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Initialize the DMA registers according to the specified parameters in DMA_InitStruct.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are initialized.
  - ERROR : Not applicable.

Notes

- This API is used for all available DMA channels.
- A software request transfer can be done once programming the direction field in memory to memory value.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_DeInit

Function name

uint32_t LL_DMA_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA registers to their default reset values.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are de-initialized.
  - ERROR : DMA registers are not de-initialized.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_StructInit

Function name

void LL_DMA_StructInit (LL_DMA_InitTypeDef * DMA_InitStruct)

Function description

Set each LL_DMA_InitTypeDef field to default value.

Parameters

- **DMA_InitStruct:** Pointer to a LL_DMA_InitTypeDef structure.

Return values

- **None.:**

LL_DMA_ListStructInit

Function name

void LL_DMA_ListStructInit (LL_DMA_InitLinkedListTypeDef * DMA_InitLinkedListStruct)

Function description

Set each LL_DMA_InitLinkedListTypeDef field to default value.

Parameters

- **DMA_InitLinkedListStruct:** Pointer to a LL_DMA_InitLinkedListTypeDef structure.

Return values

- **None.:**

LL_DMA_NodeStructInit

Function name

void LL_DMA_NodeStructInit (LL_DMA_InitNodeTypeDef * DMA_InitNodeStruct)

Function description

Set each LL_DMA_InitNodeTypeDef field to default value.

Parameters

- **DMA_InitNodeStruct:** Pointer to a LL_DMA_InitNodeTypeDef structure.

Return values

- **None.:**

LL_DMA_List_Init

Function name

uint32_t LL_DMA_List_Init (DMA_TypeDef * DMAx, uint32_t Channel, LL_DMA_InitLinkedListTypeDef * DMA_InitLinkedListStruct)

Function description

Initialize the DMA linked list according to the specified parameters in LL_DMA_InitLinkedListTypeDef.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  – LL_DMA_CHANNEL_0
  – LL_DMA_CHANNEL_1
  – LL_DMA_CHANNEL_2
  – LL_DMA_CHANNEL_3
  – LL_DMA_CHANNEL_4
  – LL_DMA_CHANNEL_5
  – LL_DMA_CHANNEL_6
  – LL_DMA_CHANNEL_7
  – LL_DMA_CHANNEL_8
  – LL_DMA_CHANNEL_9
  – LL_DMA_CHANNEL_10
  – LL_DMA_CHANNEL_11
  – LL_DMA_CHANNEL_12
  – LL_DMA_CHANNEL_13
  – LL_DMA_CHANNEL_14
  – LL_DMA_CHANNEL_15
- **DMA_InitLinkedListStruct:** pointer to a LL_DMA_InitLinkedListTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS : DMA registers are initialized.
  – ERROR : Not applicable.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_List_DeInit

Function name

uint32_t LL_DMA_List_DeInit (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

De-initialize the DMA linked list.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMA_CHANNEL_0
  - LL_DMA_CHANNEL_1
  - LL_DMA_CHANNEL_2
  - LL_DMA_CHANNEL_3
  - LL_DMA_CHANNEL_4
  - LL_DMA_CHANNEL_5
  - LL_DMA_CHANNEL_6
  - LL_DMA_CHANNEL_7
  - LL_DMA_CHANNEL_8
  - LL_DMA_CHANNEL_9
  - LL_DMA_CHANNEL_10
  - LL_DMA_CHANNEL_11
  - LL_DMA_CHANNEL_12
  - LL_DMA_CHANNEL_13
  - LL_DMA_CHANNEL_14
  - LL_DMA_CHANNEL_15

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS : DMA registers are de-initialized.
  - ERROR : DMA registers are not de-initialized.

Notes

- This API is used for all available DMA channels.
- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : LL_DMA_GET_INSTANCE LL_DMA_GET_CHANNEL

LL_DMA_CreateLinkNode

Function name

uint32_t LL_DMA_CreateLinkNode (LL_DMA_InitNodeTypeDef * DMA_InitNodeStruct, LL_DMA_LinkNodeTypeDef * pNode)

Function description

Initializes DMA linked list node according to the specified parameters in the DMA_InitNodeStruct.

Parameters

- **DMA_InitNodeStruct:** Pointer to a LL_DMA_InitNodeTypeDef structure that contains linked list node registers configurations.
- **pNode:** Pointer to linked list node to fill according to LL_DMA_LinkNodeTypeDef parameters.

Return values

- **None:**

LL_DMA_ConnectLinkNode

Function name

void LL_DMA_ConnectLinkNode (LL_DMA_LinkNodeTypeDef * pPrevLinkNode, uint32_t PrevNodeCLLRIdx, LL_DMA_LinkNodeTypeDef * pNewLinkNode, uint32_t NewNodeCLLRIdx)

Function description

Connect Linked list Nodes.

Parameters

- **pPrevLinkNode:** Pointer to previous linked list node to be connected to new Linked list node.
- **PrevNodeCLLRIdx:** Offset of Previous Node CLLR register. This parameter can be a value of CLLR offset.
- **pNewLinkNode:** Pointer to new Linked list.
- **NewNodeCLLRIdx:** Offset of New Node CLLR register. This parameter can be a value of CLLR offset.

Return values

- **None:**

LL_DMA_DisconnectNextLinkNode

Function name

void LL_DMA_DisconnectNextLinkNode (LL_DMA_LinkNodeTypeDef * pLinkNode, uint32_t LinkNodeCLLRIdx)

Function description

Disconnect the next linked list node.

Parameters

- **pLinkNode:** Pointer to linked list node to be disconnected from the next one.
- **LinkNodeCLLRIdx:** Offset of Link Node CLLR register.

Return values

- **None.:**

101.3 DMA Firmware driver defines

The following section lists the various define and macros of the module.

101.3.1 DMA

DMA

Block Hardware Request

LL_DMA_HWREQUEST_SINGLEBURST

Hardware request is driven by a peripheral with a hardware request/acknowledge protocol at a burst level

LL_DMA_HWREQUEST_BLK

Hardware request is driven by a peripheral with a hardware request/acknowledge protocol at a block level

Block Repeat Destination Address Update Mode

LL_DMA_BLKRPT_DEST_ADDR_INCREMENT

Destination address is incremented after each block transfer by destination update value

LL_DMA_BLKRPT_DEST_ADDR_DECREMENT

Destination address is decremented after each block transfer by destination update value

Block Repeat Source Address Update Mode

LL_DMA_BLKRPT_SRC_ADDR_INCREMENT

Source address pointer is incremented after each block transfer by source update value

LL_DMA_BLKRPT_SRC_ADDR_DECREMENT

Source address pointer is decremented after each block transfer by source update value

Channel

LL_DMA_CHANNEL_0

LL_DMA_CHANNEL_1

LL_DMA_CHANNEL_2

LL_DMA_CHANNEL_3

LL_DMA_CHANNEL_4

LL_DMA_CHANNEL_5

LL_DMA_CHANNEL_6

LL_DMA_CHANNEL_7

LL_DMA_CHANNEL_8

LL_DMA_CHANNEL_9

LL_DMA_CHANNEL_10

LL_DMA_CHANNEL_11

LL_DMA_CHANNEL_12

LL_DMA_CHANNEL_13

LL_DMA_CHANNEL_14

LL_DMA_CHANNEL_15

LL_DMA_CHANNEL_ALL

CLLR offset

LL_DMA_CLLR_OFFSET0

LL_DMA_CLLR_OFFSET1

LL_DMA_CLLR_OFFSET2

LL_DMA_CLLR_OFFSET3

LL_DMA_CLLR_OFFSET4

LL_DMA_CLLR_OFFSET5

LL_DMA_CLLR_OFFSET6

LL_DMA_CLLR_OFFSET7

Data Alignment

LL_DMA_DATA_ALIGN_ZEROPADD

If src data width < dest data width : => Right Aligned padded with 0 up to destination data width. If src data width > dest data width : => Right Aligned Left Truncated down to destination data width.

LL_DMA_DATA_ALIGN_SIGNEXTPADD

If src data width < dest data width : => Right Aligned padded with sign extended up to destination data width. If src data width > dest data width : => Left Aligned Right Truncated down to the destination data width

LL_DMA_DATA_PACK_UNPACK

If src data width < dest data width : => Packed at the destination data width (Not Available for LPDMA) If src data width > dest data width : => Unpacked at the destination data width (Not Available for LPDMA)

Destination Allocated Port

LL_DMA_DEST_ALLOCATED_PORT0

Destination Allocated Port 0

LL_DMA_DEST_ALLOCATED_PORT1

Destination Allocated Port 1

Destination Data Width

LL_DMA_DEST_DATAWIDTH_BYTE

Destination Data Width : Byte

LL_DMA_DEST_DATAWIDTH_HALFWORD

Destination Data Width : HalfWord

LL_DMA_DEST_DATAWIDTH_WORD

Destination Data Width : Word

Destination Increment Mode

LL_DMA_DEST_FIXED

Destination fixed single/burst

LL_DMA_DEST_INCREMENT

Destination incremented single/burst

Destination Security Attribute

LL_DMA_CHANNEL_DEST_NSEC

NSecure transfer from the destination

LL_DMA_CHANNEL_DEST_SEC

Secure transfer from the destination

Burst Destination Address Update Mode

LL_DMA_BURST_DEST_ADDR_INCREMENT

Destination address pointer is incremented after each burst transfer by destination update value

LL_DMA_BURST_DEST_ADDR_DECREMENT

Destination address pointer is decremented after each burst transfer by destination update value

Destination Byte Exchange

LL_DMA_DEST_BYTE_PRESERVE

No destination Byte exchange when destination data width > Byte

LL_DMA_DEST_BYTE_EXCHANGE

Destination Byte exchange when destination data width > Byte

Destination Half-Word Exchange

LL_DMA_DEST_HALFWORD_PRESERVE

No destination Half-Word exchange when destination data width is word

LL_DMA_DEST_HALFWORD_EXCHANGE

Destination Half-Word exchange when destination data width is word

Linked list node type

LL_DMA_LPDMA_LINEAR_NODE

LPDMA node : linear addressing node

LL_DMA_GPDMA_LINEAR_NODE

GPDMA node : linear addressing node

LL_DMA_GPDMA_2D_NODE

GPDMA node : 2 dimension addressing node

Linked list register update

LL_DMA_UPDATE_CTR1

Update CTR1 register from memory : available for all DMA channels

LL_DMA_UPDATE_CTR2

Update CTR2 register from memory : available for all DMA channels

LL_DMA_UPDATE_CBR1

Update CBR1 register from memory : available for all DMA channels

LL_DMA_UPDATE_CSAR

Update CSAR register from memory : available for all DMA channels

LL_DMA_UPDATE_CDAR

Update CDAR register from memory : available for all DMA channels

LL_DMA_UPDATE_CTR3

Update CTR3 register from memory : available only for 2D addressing DMA channels

LL_DMA_UPDATE_CBR2

Update CBR2 register from memory : available only for 2D addressing DMA channels

LL_DMA_UPDATE_CLLR

Update CLLR register from memory : available for all DMA channels

Linked List Allocated Port

LL_DMA_LINK_ALLOCATED_PORT0

Linked List Allocated Port 0

LL_DMA_LINK_ALLOCATED_PORT1

Linked List Allocated Port 1

Link Step Mode

LL_DMA_LSM_FULL_EXECUTION

Channel execute the full linked list

LL_DMA_LSM_1LINK_EXECUTION

Channel execute one node of the linked list

Priority Level

LL_DMA_LOW_PRIORITY_LOW_WEIGHT

Priority level : Low Priority, Low Weight

LL_DMA_LOW_PRIORITY_MID_WEIGHT

Priority level : Low Priority, Mid Weight

LL_DMA_LOW_PRIORITY_HIGH_WEIGHT

Priority level : Low Priority, High Weight

LL_DMA_HIGH_PRIORITY

Priority level : High Priority

Request Selection

LL_GPDMA1_REQUEST_ADC1

GPDMA1 HW Request is ADC1

LL_GPDMA1_REQUEST_ADC4

GPDMA1 HW Request is ADC4

LL_GPDMA1_REQUEST_DAC1_CH1

GPDMA1 HW Request is DAC1_CH1

LL_GPDMA1_REQUEST_DAC1_CH2

GPDMA1 HW Request is DAC1_CH2

LL_GPDMA1_REQUEST_TIM6_UP

GPDMA1 HW Request is TIM6_UP

LL_GPDMA1_REQUEST_TIM7_UP

GPDMA1 HW Request is TIM7_UP

LL_GPDMA1_REQUEST_SPI1_RX

GPDMA1 HW Request is SPI1_RX

LL_GPDMA1_REQUEST_SPI1_TX
GPDMA1 HW Request is SPI1_TX

LL_GPDMA1_REQUEST_SPI2_RX
GPDMA1 HW Request is SPI2_RX

LL_GPDMA1_REQUEST_SPI2_TX
GPDMA1 HW Request is SPI2_TX

LL_GPDMA1_REQUEST_SPI3_RX
GPDMA1 HW Request is SPI3_RX

LL_GPDMA1_REQUEST_SPI3_TX
GPDMA1 HW Request is SPI3_TX

LL_GPDMA1_REQUEST_I2C1_RX
GPDMA1 HW Request is I2C1_RX

LL_GPDMA1_REQUEST_I2C1_TX
GPDMA1 HW Request is I2C1_TX

LL_GPDMA1_REQUEST_I2C1_EVC
GPDMA1 HW Request is I2C1_EVC

LL_GPDMA1_REQUEST_I2C2_RX
GPDMA1 HW Request is I2C2_RX

LL_GPDMA1_REQUEST_I2C2_TX
GPDMA1 HW Request is I2C2_TX

LL_GPDMA1_REQUEST_I2C2_EVC
GPDMA1 HW Request is I2C2_EVC

LL_GPDMA1_REQUEST_I2C3_RX
GPDMA1 HW Request is I2C3_RX

LL_GPDMA1_REQUEST_I2C3_TX
GPDMA1 HW Request is I2C3_TX

LL_GPDMA1_REQUEST_I2C3_EVC
GPDMA1 HW Request is I2C3_EVC

LL_GPDMA1_REQUEST_I2C4_RX
GPDMA1 HW Request is I2C4_RX

LL_GPDMA1_REQUEST_I2C4_TX
GPDMA1 HW Request is I2C4_TX

LL_GPDMA1_REQUEST_I2C4_EVC
GPDMA1 HW Request is I2C4_EVC

LL_GPDMA1_REQUEST_USART1_RX
GPDMA1 HW Request is USART1_RX

LL_GPDMA1_REQUEST_USART1_TX
GPDMA1 HW Request is USART1_TX

LL_GPDMA1_REQUEST_USART2_RX
GPDMA1 HW Request is USART2_RX

LL_GPDMA1_REQUEST_USART2_TX
GPDMA1 HW Request is USART2_TX

LL_GPDMA1_REQUEST_USART3_RX
GPDMA1 HW Request is USART3_RX

LL_GPDMA1_REQUEST_USART3_TX
GPDMA1 HW Request is USART3_TX

LL_GPDMA1_REQUEST_UART4_RX
GPDMA1 HW Request is UART4_RX

LL_GPDMA1_REQUEST_UART4_TX
GPDMA1 HW Request is UART4_TX

LL_GPDMA1_REQUEST_UART5_RX
GPDMA1 HW Request is UART5_RX

LL_GPDMA1_REQUEST_UART5_TX
GPDMA1 HW Request is UART5_TX

LL_GPDMA1_REQUEST_LPUART1_RX
GPDMA1 HW Request is LPUART1_RX

LL_GPDMA1_REQUEST_LPUART1_TX
GPDMA1 HW Request is LPUART1_TX

LL_GPDMA1_REQUEST_SAI1_A
GPDMA1 HW Request is SAI1_A

LL_GPDMA1_REQUEST_SAI1_B
GPDMA1 HW Request is SAI1_B

LL_GPDMA1_REQUEST_SAI2_A
GPDMA1 HW Request is SAI2_A

LL_GPDMA1_REQUEST_SAI2_B
GPDMA1 HW Request is SAI2_B

LL_GPDMA1_REQUEST_OCTOSPI1
GPDMA1 HW Request is OCTOSPI1

LL_GPDMA1_REQUEST_OCTOSPI2
GPDMA1 HW Request is OCTOSPI2

LL_GPDMA1_REQUEST_TIM1_CH1
GPDMA1 HW Request is TIM1_CH1

LL_GPDMA1_REQUEST_TIM1_CH2
GPDMA1 HW Request is TIM1_CH2

LL_GPDMA1_REQUEST_TIM1_CH3
GPDMA1 HW Request is TIM1_CH3

LL_GPDMA1_REQUEST_TIM1_CH4
GPDMA1 HW Request is TIM1_CH4

LL_GPDMA1_REQUEST_TIM1_UP
GPDMA1 HW Request is TIM1_UP

LL_GPDMA1_REQUEST_TIM1_TRIG
GPDMA1 HW Request is TIM1_TRIG

LL_GPDMA1_REQUEST_TIM1_COM
GPDMA1 HW Request is TIM1_COM

LL_GPDMA1_REQUEST_TIM8_CH1
GPDMA1 HW Request is TIM8_CH1

LL_GPDMA1_REQUEST_TIM8_CH2
GPDMA1 HW Request is TIM8_CH2

LL_GPDMA1_REQUEST_TIM8_CH3
GPDMA1 HW Request is TIM8_CH3

LL_GPDMA1_REQUEST_TIM8_CH4
GPDMA1 HW Request is TIM8_CH4

LL_GPDMA1_REQUEST_TIM8_UP
GPDMA1 HW Request is TIM8_UP

LL_GPDMA1_REQUEST_TIM8_TRIG
GPDMA1 HW Request is TIM8_TRIG

LL_GPDMA1_REQUEST_TIM8_COM
GPDMA1 HW Request is TIM8_COM

LL_GPDMA1_REQUEST_TIM2_CH1
GPDMA1 HW Request is TIM2_CH1

LL_GPDMA1_REQUEST_TIM2_CH2
GPDMA1 HW Request is TIM2_CH2

LL_GPDMA1_REQUEST_TIM2_CH3
GPDMA1 HW Request is TIM2_CH3

LL_GPDMA1_REQUEST_TIM2_CH4
GPDMA1 HW Request is TIM2_CH4

LL_GPDMA1_REQUEST_TIM2_UP
GPDMA1 HW Request is TIM2_UP

LL_GPDMA1_REQUEST_TIM3_CH1
GPDMA1 HW Request is TIM3_CH1

LL_GPDMA1_REQUEST_TIM3_CH2
GPDMA1 HW Request is TIM3_CH2

LL_GPDMA1_REQUEST_TIM3_CH3
GPDMA1 HW Request is TIM3_CH3

LL_GPDMA1_REQUEST_TIM3_CH4
GPDMA1 HW Request is TIM3_CH4

LL_GPDMA1_REQUEST_TIM3_UP
GPDMA1 HW Request is TIM3_UP

LL_GPDMA1_REQUEST_TIM3_TRIG
GPDMA1 HW Request is TIM3_TRIG

LL_GPDMA1_REQUEST_TIM4_CH1
GPDMA1 HW Request is TIM4_CH1

LL_GPDMA1_REQUEST_TIM4_CH2
GPDMA1 HW Request is TIM4_CH2

LL_GPDMA1_REQUEST_TIM4_CH3
GPDMA1 HW Request is TIM4_CH3

LL_GPDMA1_REQUEST_TIM4_CH4
GPDMA1 HW Request is TIM4_CH4

LL_GPDMA1_REQUEST_TIM4_UP
GPDMA1 HW Request is TIM4_UP

LL_GPDMA1_REQUEST_TIM5_CH1
GPDMA1 HW Request is TIM5_CH1

LL_GPDMA1_REQUEST_TIM5_CH2
GPDMA1 HW Request is TIM5_CH2

LL_GPDMA1_REQUEST_TIM5_CH3
GPDMA1 HW Request is TIM5_CH3

LL_GPDMA1_REQUEST_TIM5_CH4
GPDMA1 HW Request is TIM5_CH4

LL_GPDMA1_REQUEST_TIM5_UP
GPDMA1 HW Request is TIM5_UP

LL_GPDMA1_REQUEST_TIM5_TRIG
GPDMA1 HW Request is TIM5_TRIG

LL_GPDMA1_REQUEST_TIM15_CH1
GPDMA1 HW Request is TIM15_CH1

LL_GPDMA1_REQUEST_TIM15_UP
GPDMA1 HW Request is TIM15_UP

LL_GPDMA1_REQUEST_TIM15_TRIG
GPDMA1 HW Request is TIM15_TRIG

LL_GPDMA1_REQUEST_TIM15_COM
GPDMA1 HW Request is TIM15_COM

LL_GPDMA1_REQUEST_TIM16_CH1
GPDMA1 HW Request is TIM16_CH1

LL_GPDMA1_REQUEST_TIM16_UP
GPDMA1 HW Request is TIM16_UP

LL_GPDMA1_REQUEST_TIM17_CH1
GPDMA1 HW Request is TIM17_CH1

LL_GPDMA1_REQUEST_TIM17_UP
GPDMA1 HW Request is TIM17_UP

LL_GPDMA1_REQUEST_DCMI_PSSI
GPDMA1 HW Request is DCMI_PSSI

LL_GPDMA1_REQUEST_AES_IN
GPDMA1 HW Request is AES_IN

LL_GPDMA1_REQUEST_AES_OUT
GPDMA1 HW Request is AES_OUT

LL_GPDMA1_REQUEST_HASH_IN
GPDMA1 HW Request is HASH_IN

LL_GPDMA1_REQUEST_UCPD1_TX
GPDMA1 HW Request is UCPD1_TX

LL_GPDMA1_REQUEST_UCPD1_RX
GPDMA1 HW Request is UCPD1_RX

LL_GPDMA1_REQUEST_MDF1_FLT0
GPDMA1 HW Request is MDF1_FLT0

LL_GPDMA1_REQUEST_MDF1_FLT1
GPDMA1 HW Request is MDF1_FLT1

LL_GPDMA1_REQUEST_MDF1_FLT2
GPDMA1 HW Request is MDF1_FLT2

LL_GPDMA1_REQUEST_MDF1_FLT3
GPDMA1 HW Request is MDF1_FLT3

LL_GPDMA1_REQUEST_MDF1_FLT4
GPDMA1 HW Request is MDF1_FLT4

LL_GPDMA1_REQUEST_MDF1_FLT5
GPDMA1 HW Request is MDF1_FLT5

LL_GPDMA1_REQUEST_ADF1_FLT0
GPDMA1 HW Request is ADF1_FLT0

LL_GPDMA1_REQUEST_FMAC_READ
GPDMA1 HW Request is FMAC_READ

LL_GPDMA1_REQUEST_FMAC_WRITE
GPDMA1 HW Request is FMAC_WRITE

LL_GPDMA1_REQUEST_CORDIC_READ
GPDMA1 HW Request is CORDIC_READ

LL_GPDMA1_REQUEST_CORDIC_WRITE
GPDMA1 HW Request is CORDIC_WRITE

LL_GPDMA1_REQUEST_SAES_IN
GPDMA1 HW Request is SAES_IN

LL_GPDMA1_REQUEST_SAES_OUT
GPDMA1 HW Request is SAES_OUT

LL_GPDMA1_REQUEST_LPTIM1_IC1
GPDMA1 HW Request is LPTIM1_IC1

LL_GPDMA1_REQUEST_LPTIM1_IC2
GPDMA1 HW Request is LPTIM1_IC2

LL_GPDMA1_REQUEST_LPTIM1_UE
GPDMA1 HW Request is LPTIM1_UE

LL_GPDMA1_REQUEST_LPTIM2_IC1
GPDMA1 HW Request is LPTIM2_IC1

LL_GPDMA1_REQUEST_LPTIM2_IC2
GPDMA1 HW Request is LPTIM2_IC2

LL_GPDMA1_REQUEST_LPTIM2_UE
GPDMA1 HW Request is LPTIM2_UE

LL_GPDMA1_REQUEST_LPTIM3_IC1
GPDMA1 HW Request is LPTIM3_IC1

LL_GPDMA1_REQUEST_LPTIM3_IC2
GPDMA1 HW Request is LPTIM3_IC2

LL_GPDMA1_REQUEST_LPTIM3_UE
GPDMA1 HW Request is LPTIM3_UE

LL_GPDMA1_REQUEST_HSPI1
GPDMA1 HW request is HSPI1

LL_GPDMA1_REQUEST_I2C5_RX
GPDMA1 HW request is I2C5_RX

LL_GPDMA1_REQUEST_I2C5_TX
GPDMA1 HW request is I2C5_TX

LL_GPDMA1_REQUEST_I2C5_EVC
GPDMA1 HW request is I2C5_EVC

LL_GPDMA1_REQUEST_I2C6_RX
GPDMA1 HW request is I2C6_RX

LL_GPDMA1_REQUEST_I2C6_TX
GPDMA1 HW request is I2C6_TX

LL_GPDMA1_REQUEST_I2C6_EVC
GPDMA1 HW request is I2C6_EVC

LL_GPDMA1_REQUEST_USART6_RX

GPDMA1 HW request is USART6_RX

LL_GPDMA1_REQUEST_USART6_TX

GPDMA1 HW request is USART6_TX

LL_GPDMA1_REQUEST_ADC2

GPDMA1 HW request is ADC2

LL_GPDMA1_REQUEST_JPEG_RX

GPDMA1 HW request is JPEG_TX

LL_GPDMA1_REQUEST_JPEG_TX

GPDMA1 HW request is JPEG_RX

LL_GPDMA1_REQUEST_DCMI

LL_LPDMA1_REQUEST_LPUART1_RX

LPDMA1 HW Request is LPUART1_RX

LL_LPDMA1_REQUEST_LPUART1_TX

LPDMA1 HW Request is LPUART1_TX

LL_LPDMA1_REQUEST_SPI3_RX

LPDMA1 HW Request is SPI3_RX

LL_LPDMA1_REQUEST_SPI3_TX

LPDMA1 HW Request is SPI3_TX

LL_LPDMA1_REQUEST_I2C3_RX

LPDMA1 HW Request is I2C3_RX

LL_LPDMA1_REQUEST_I2C3_TX

LPDMA1 HW Request is I2C3_TX

LL_LPDMA1_REQUEST_I2C3_EVC

LPDMA1 HW Request is I2C3_EVC

LL_LPDMA1_REQUEST_ADC4

LPDMA1 HW Request is ADC4

LL_LPDMA1_REQUEST_DAC1_CH1

LPDMA1 HW Request is DAC1_CH1

LL_LPDMA1_REQUEST_DAC1_CH2

LPDMA1 HW Request is DAC1_CH2

LL_LPDMA1_REQUEST_ADF1_FLT0

LPDMA1 HW Request is ADF1_FLT0

LL_LPDMA1_REQUEST_LPTIM1_IC1

LPDMA1 HW Request is LPTIM1_IC1

LL_LPDMA1_REQUEST_LPTIM1_IC2

LPDMA1 HW Request is LPTIM1_IC2

LL_LPDMA1_REQUEST_LPTIM1_UE

LPDMA1 HW Request is LPTIM1_UE

LL_LPDMA1_REQUEST_LPTIM3_IC1

LPDMA1 HW Request is LPTIM3_IC1

LL_LPDMA1_REQUEST_LPTIM3_IC2

LPDMA1 HW Request is LPTIM3_IC2

LL_LPDMA1_REQUEST_LPTIM3_UE

LPDMA1 HW Request is LPTIM3_UE

Source Allocated Port

LL_DMA_SRC_ALLOCATED_PORT0

Source Allocated Port 0

LL_DMA_SRC_ALLOCATED_PORT1

Source Allocated Port 1

Source Data Width

LL_DMA_SRC_DATAWIDTH_BYTE

Source Data Width : Byte

LL_DMA_SRC_DATAWIDTH_HALFWORD

Source Data Width : HalfWord

LL_DMA_SRC_DATAWIDTH_WORD

Source Data Width : Word

Source Increment Mode

LL_DMA_SRC_FIXED

Source fixed single/burst

LL_DMA_SRC_INCREMENT

Source incremented single/burst

Source Security Attribute

LL_DMA_CHANNEL_NSEC

NSecure channel

LL_DMA_CHANNEL_SEC

Secure channel

LL_DMA_CHANNEL_SRC_NSEC

NSecure transfer from the source

LL_DMA_CHANNEL_SRC_SEC

Secure transfer from the source

Burst Source Address Update Mode

LL_DMA_BURST_SRC_ADDR_INCREMENT

Source address pointer is incremented after each burst transfer by source update value

LL_DMA_BURST_SRC_ADDR_DECREMENT

Source address pointer is decremented after each burst transfer by source update value

Source Byte Exchange

LL_DMA_SRC_BYTE_PRESERVE

No source Byte exchange when source data width is word

LL_DMA_SRC_BYTE_EXCHANGE

Source Byte exchange when source data width is word

Transfer Direction

LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_DMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

Transfer Event Mode

LL_DMA_TCEM_BLK_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of each block

LL_DMA_TCEM_RPT_BLK_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of the repeated block

LL_DMA_TCEM_EACH_LLITEM_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of each linked-list item

LL_DMA_TCEM_LAST_LLITEM_TRANSFER

The TC (and the HT) event is generated at the (respectively half) end of the last linked-list item

Transfer Trigger Mode

LL_DMA_TRIGM_BLK_TRANSFER

A block transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_RPT_BLK_TRANSFER

A repeated block transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_LLI_LINK_TRANSFER

A LLI link transfer is conditioned by (at least) one hit trigger

LL_DMA_TRIGM_SINGLBURST_TRANSFER

A Single/Burst transfer is conditioned by (at least) one hit trigger

Trigger Polarity

LL_DMA_TRIG_POLARITY_MASKED

No trigger of the selected DMA request. Masked trigger event

LL_DMA_TRIG_POLARITY_RISING

Trigger of the selected DMA request on the rising edge of the selected trigger event input

LL_DMA_TRIG_POLARITY_FALLING

Trigger of the selected DMA request on the falling edge of the selected trigger event input

Trigger Selection

LL_GPDMA1_TRIGGER_EXTI_LINE0
GPDMA1 HW Trigger is EXTI_LINE0

LL_GPDMA1_TRIGGER_EXTI_LINE1
GPDMA1 HW Trigger is EXTI_LINE1

LL_GPDMA1_TRIGGER_EXTI_LINE2
GPDMA1 HW Trigger is EXTI_LINE2

LL_GPDMA1_TRIGGER_EXTI_LINE3
GPDMA1 HW Trigger is EXTI_LINE3

LL_GPDMA1_TRIGGER_EXTI_LINE4
GPDMA1 HW Trigger is EXTI_LINE4

LL_GPDMA1_TRIGGER_EXTI_LINE5
GPDMA1 HW Trigger is EXTI_LINE5

LL_GPDMA1_TRIGGER_EXTI_LINE6
GPDMA1 HW Trigger is EXTI_LINE6

LL_GPDMA1_TRIGGER_EXTI_LINE7
GPDMA1 HW Trigger is EXTI_LINE7

LL_GPDMA1_TRIGGER_TAMP_TRG1
GPDMA1 HW Trigger is TAMP_TRG1

LL_GPDMA1_TRIGGER_TAMP_TRG2
GPDMA1 HW Trigger is TAMP_TRG2

LL_GPDMA1_TRIGGER_TAMP_TRG3
GPDMA1 HW Trigger is TAMP_TRG3

LL_GPDMA1_TRIGGER_LPTIM1_CH1
GPDMA1 HW Trigger is LPTIM1_CH1

LL_GPDMA1_TRIGGER_LPTIM1_CH2
GPDMA1 HW Trigger is LPTIM1_CH2

LL_GPDMA1_TRIGGER_LPTIM2_CH1
GPDMA1 HW Trigger is LPTIM2_CH1

LL_GPDMA1_TRIGGER_LPTIM2_CH2
GPDMA1 HW Trigger is LPTIM2_CH2

LL_GPDMA1_TRIGGER_LPTIM4_OUT
GPDMA1 HW Trigger is LPTIM4_OUT

LL_GPDMA1_TRIGGER_COMP1_OUT
GPDMA1 HW Trigger is COMP1_OUT

LL_GPDMA1_TRIGGER_COMP2_OUT
GPDMA1 HW Trigger is COMP2_OUT

LL_GPDMA1_TRIGGER_RTC_ALRA_TRG
GPDMA1 HW Trigger is RTC_ALRA_TRG

LL_GPDMA1_TRIGGER_RTC_ALRB_TRG
GPDMA1 HW Trigger is RTC_ALRB_TRG

LL_GPDMA1_TRIGGER_RTC_WUT_TRG
GPDMA1 HW Trigger is RTC_WUT_TRG

LL_GPDMA1_TRIGGER_GPDMA1_CH0_TCF
GPDMA1 HW Trigger is GPDMA1_CH0_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH1_TCF
GPDMA1 HW Trigger is GPDMA1_CH1_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH2_TCF
GPDMA1 HW Trigger is GPDMA1_CH2_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH3_TCF
GPDMA1 HW Trigger is GPDMA1_CH3_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH4_TCF
GPDMA1 HW Trigger is GPDMA1_CH4_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH5_TCF
GPDMA1 HW Trigger is GPDMA1_CH5_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH6_TCF
GPDMA1 HW Trigger is GPDMA1_CH6_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH7_TCF
GPDMA1 HW Trigger is GPDMA1_CH7_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH8_TCF
GPDMA1 HW Trigger is GPDMA1_CH8_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH9_TCF
GPDMA1 HW Trigger is GPDMA1_CH9_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH10_TCF
GPDMA1 HW Trigger is GPDMA1_CH10_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH11_TCF
GPDMA1 HW Trigger is GPDMA1_CH11_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH12_TCF
GPDMA1 HW Trigger is GPDMA1_CH12_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH13_TCF
GPDMA1 HW Trigger is GPDMA1_CH13_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH14_TCF
GPDMA1 HW Trigger is GPDMA1_CH14_TCF

LL_GPDMA1_TRIGGER_GPDMA1_CH15_TCF
GPDMA1 HW Trigger is GPDMA1_CH15_TCF

LL_GPDMA1_TRIGGER_LPDMA1_CH0_TCF
GPDMA1 HW Trigger is LPDMA1_CH0_TCF

LL_GPDMA1_TRIGGER_LPDMA1_CH1_TCF
GPDMA1 HW Trigger is LPDMA1_CH1_TCF

LL_GPDMA1_TRIGGER_LPDMA1_CH2_TCF
GPDMA1 HW Trigger is LPDMA1_CH2_TCF

LL_GPDMA1_TRIGGER_LPDMA1_CH3_TCF
GPDMA1 HW Trigger is LPDMA1_CH3_TCF

LL_GPDMA1_TRIGGER_TIM2_TRGO
GPDMA1 HW Trigger is TIM2_TRGO

LL_GPDMA1_TRIGGER_TIM15_TRGO
GPDMA1 HW Trigger is TIM15_TRGO

LL_GPDMA1_TRIGGER_ADC4_AWD1
GPDMA1 HW Trigger is ADC4_AWD1

LL_GPDMA1_TRIGGER_ADC1_AWD1
GPDMA1 HW Trigger is ADC1_AWD1

LL_GPDMA1_TRIGGER_TIM3_TRGO
GPDMA1 HW Trigger signal is TIM3_TRGO

LL_GPDMA1_TRIGGER_TIM4_TRGO
GPDMA1 HW Trigger signal is TIM4_TRGO

LL_GPDMA1_TRIGGER_TIM5_TRGO
GPDMA1 HW Trigger signal is TIM5_TRGO

LL_GPDMA1_TRIGGER_LTDC_LI
GPDMA1 HW Trigger signal is LTDC_LI

LL_GPDMA1_TRIGGER_DSI_TE
GPDMA1 HW Trigger signal is DSI_TE

LL_GPDMA1_TRIGGER_DSI_ER
GPDMA1 HW Trigger signal is DSI_ER

LL_GPDMA1_TRIGGER_DMA2D_TC
GPDMA1 HW Trigger signal is DMA2D_TC

LL_GPDMA1_TRIGGER_DMA2D_CTC
GPDMA1 HW Trigger signal is DMA2D_CTC

LL_GPDMA1_TRIGGER_DMA2D_TW
GPDMA1 HW Trigger signal is DMA2D_TW

LL_GPDMA1_TRIGGER_GPU2D_FLAG0
GPDMA1 HW Trigger signal is GPU2D_FLAG0

LL_GPDMA1_TRIGGER_GPU2D_FLAG1
GPDMA1 HW Trigger signal is GPU2D_FLAG1

LL_GPDMA1_TRIGGER_GPU2D_FLAG2
GPDMA1 HW Trigger signal is GPU2D_FLAG2

LL_GPDMA1_TRIGGER_GPU2D_FLAG3
GPDMA1 HW Trigger signal is GPU2D_FLAG3

LL_GPDMA1_TRIGGER_GFXTIM_EVT3
GPDMA1 HW Trigger signal is GFXTIM_EVT3

LL_GPDMA1_TRIGGER_GFXTIM_EVT2
GPDMA1 HW Trigger signal is GFXTIM_EVT2

LL_GPDMA1_TRIGGER_GFXTIM_EVT1
GPDMA1 HW Trigger signal is GFXTIM_EVT1

LL_GPDMA1_TRIGGER_GFXTIM_EVT0
GPDMA1 HW Trigger signal is GFXTIM_EVT0

LL_GPDMA1_TRIGGER_JPEG_EOC
GPDMA1 HW Trigger signal is JPEG_EOC

LL_GPDMA1_TRIGGER_JPEG_IFNF
GPDMA1 HW Trigger signal is JPEG_IFNF

LL_GPDMA1_TRIGGER_JPEG_IFT
GPDMA1 HW Trigger signal is JPEG_IFT

LL_GPDMA1_TRIGGER_JPEG_OFNE
GPDMA1 HW Trigger signal is JPEG_OFNE

LL_GPDMA1_TRIGGER_JPEG_OFT
GPDMA1 HW Trigger signal is JPEG_OFT

LL_LPDMA1_TRIGGER_EXTI_LINE0
LPDMA1 HW Trigger is EXTI_LINE0

LL_LPDMA1_TRIGGER_EXTI_LINE1
LPDMA1 HW Trigger is EXTI_LINE1

LL_LPDMA1_TRIGGER_EXTI_LINE2
LPDMA1 HW Trigger is EXTI_LINE2

LL_LPDMA1_TRIGGER_EXTI_LINE3
LPDMA1 HW Trigger is EXTI_LINE3

LL_LPDMA1_TRIGGER_EXTI_LINE4
LPDMA1 HW Trigger is EXTI_LINE4

LL_LPDMA1_TRIGGER_TAMP_TRG1
LPDMA1 HW Trigger is TAMP_TRG1

LL_LPDMA1_TRIGGER_TAMP_TRG2
LPDMA1 HW Trigger is TAMP_TRG2

LL_LPDMA1_TRIGGER_TAMP_TRG3
LPDMA1 HW Trigger is TAMP_TRG3

LL_LPDMA1_TRIGGER_LPTIM1_CH1
LPDMA1 HW Trigger is LPTIM1_CH1

LL_LPDMA1_TRIGGER_LPTIM1_CH2
LPDMA1 HW Trigger is LPTIM1_CH2

LL_LPDMA1_TRIGGER_LPTIM3_CH1
LPDMA1 HW Trigger is LPTIM3_CH1

LL_LPDMA1_TRIGGER_LPTIM4_OUT
LPDMA1 HW Trigger is LPTIM4_OUT

LL_LPDMA1_TRIGGER_COMP1_OUT
LPDMA1 HW Trigger is COMP1_OUT

LL_LPDMA1_TRIGGER_COMP2_OUT
LPDMA1 HW Trigger is COMP2_OUT

LL_LPDMA1_TRIGGER_RTC_ALRA_TRG
LPDMA1 HW Trigger is RTC_ALRA_TRG

LL_LPDMA1_TRIGGER_RTC_ALRB_TRG
LPDMA1 HW Trigger is RTC_ALRB_TRG

LL_LPDMA1_TRIGGER_RTC_WUT_TRG
LPDMA1 HW Trigger is RTC_WUT_TRG

LL_LPDMA1_TRIGGER_ADC4_AWD1
LPDMA1 HW Trigger is ADC4_AWD1

LL_LPDMA1_TRIGGER_LPDMA1_CH0_TCF
LPDMA1 HW Trigger is LPDMA1_CH0_TCF

LL_LPDMA1_TRIGGER_LPDMA1_CH1_TCF
LPDMA1 HW Trigger is LPDMA1_CH1_TCF

LL_LPDMA1_TRIGGER_LPDMA1_CH2_TCF
LPDMA1 HW Trigger is LPDMA1_CH2_TCF

LL_LPDMA1_TRIGGER_LPDMA1_CH3_TCF
LPDMA1 HW Trigger is LPDMA1_CH3_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH0_TCF
LPDMA1 HW Trigger is GPDMA1_CH0_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH1_TCF
LPDMA1 HW Trigger is GPDMA1_CH1_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH4_TCF
LPDMA1 HW Trigger is GPDMA1_CH4_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH5_TCF
LPDMA1 HW Trigger is GPDMA1_CH5_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH6_TCF
LPDMA1 HW Trigger is GPDMA1_CH6_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH7_TCF
LPDMA1 HW Trigger is GPDMA1_CH7_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH12_TCF

LPDMA1 HW Trigger is GPDMA1_CH12_TCF

LL_LPDMA1_TRIGGER_GPDMA1_CH13_TCF

LPDMA1 HW Trigger is GPDMA1_CH13_TCF

LL_LPDMA1_TRIGGER_TIM2_TRGO

LPDMA1 HW Trigger is TIM2_TRGO

LL_LPDMA1_TRIGGER_TIM15_TRGO

LPDMA1 HW Trigger is TIM15_TRGO

Common Write and Read Registers macros

LL_DMA_WriteReg

Description:

- Write a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance.
- __REG__: Register to be written.
- __VALUE__: Value to be written in the register.

Return value:

- None.

LL_DMA_ReadReg

Description:

- Read a value in DMA register.

Parameters:

- __INSTANCE__: DMA Instance.
- __REG__: Register to be read.

Return value:

- Register: value.

Convert DMAxChannely

LL_DMA_GET_INSTANCE

Description:

- Convert DMAx_Channely into DMAx.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely.

Return value:

- DMAx.

LL_DMA_GET_CHANNEL

Description:

- Convert DMAx_Channely into LL_DMA_CHANNEL_y.

Parameters:

- __CHANNEL_INSTANCE__: DMAx_Channely.

Return value:

- LL_DMA_CHANNEL_y.

LL_DMA_GET_CHANNEL_INSTANCE

Description:

- Convert DMA Instance DMAx and LL_DMA_CHANNEL_y into DMAx_Channely.

Parameters:

- __DMA_INSTANCE__: DMAx.
- __CHANNEL__: LL_DMA_CHANNEL_y.

Return value:

- DMAx_Channely.

102 LL EXTI Generic Driver

102.1 EXTI Firmware driver registers structures

102.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the stm32u5xx_ll_exti.h

Data Fields

- ***uint32_t Line_0_31***
- ***FunctionalState LineCommand***
- ***uint8_t Mode***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_EXTI_InitTypeDef::Line_0_31***
  Specifies the EXTI lines to be enabled or disabled for Lines in range 0 to 31 This parameter can be any combination of ***EXTI_LL_EC_LINE***
- ***FunctionalState LL_EXTI_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_EXTI_InitTypeDef::Mode***
  Specifies the mode for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_MODE***.
- ***uint8_t LL_EXTI_InitTypeDef::Trigger***
  Specifies the trigger signal active edge for the EXTI lines. This parameter can be a value of ***EXTI_LL_EC_TRIGGER***.

102.2 EXTI Firmware driver API description

The following section lists the various functions of the EXTI library.

102.2.1 Detailed description of functions

LL_EXTI_EnableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableIT_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_EnableIT_0_31

LL_EXTI_DisableIT_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableIT_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Interrupt request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_DisableIT_0_31

LL_EXTI_IsEnabledIT_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Interrupt request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)
  – LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- The reset value for the direct or internal lines (see RM) is set to 1 in order to enable the interrupt by default. Bits are set automatically at Power on.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IMR1 IMx LL_EXTI_IsEnabledIT_0_31

LL_EXTI_EnableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableEvent_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_EnableEvent_0_31

LL_EXTI_DisableEvent_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableEvent_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Event request for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_DisableEvent_0_31

LL_EXTI_IsEnabledEvent_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEvent_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Event request is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)
  – LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- EMR1 EMx LL_EXTI_IsEnabledEvent_0_31

LL_EXTI_EnableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_EnableRisingTrig_0_31

LL_EXTI_DisableRisingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a rising edge on a configurable interrupt line occurs during a write operation in the EXTI_RTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_DisableRisingTrig_0_31

LL_EXTI_IsEnabledRisingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RTSR1 RTx LL_EXTI_IsEnabledRisingTrig_0_31

LL_EXTI_EnableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_EnableFallingTrig_0_31

LL_EXTI_DisableFallingTrig_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- The configurable wakeup lines are edge-triggered. No glitch must be generated on these lines. If a Falling edge on a configurable interrupt line occurs during a write operation in the EXTI_FTSR register, the pending bit is not set. Rising and falling edge triggers can be set for the same interrupt line. In this case, both generate a trigger condition.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_DisableFallingTrig_0_31

LL_EXTI_IsEnabledFallingTrig_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledFallingTrig_0_31 (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FTSR1 FTx LL_EXTI_IsEnabledFallingTrig_0_31

LL_EXTI_GenerateSWI_0_31

Function name

__STATIC_INLINE void LL_EXTI_GenerateSWI_0_31 (uint32_t ExtiLine)

Function description

Generate a software Interrupt Event for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- If the interrupt is enabled on this line in the EXTI_IMR, writing a 1 to this bit when it is at '0' sets the corresponding pending bit in EXTI_PR resulting in an interrupt request generation. This bit is cleared by clearing the corresponding bit in the EXTI_PR register (by writing a 1 into the bit)
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SWIER1 SWIx LL_EXTI_GenerateSWI_0_31

LL_EXTI_IsActiveFallingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Falling Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_IsActiveFallingFlag_0_31

LL_EXTI_ReadFallingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Falling Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_ReadFallingFlag_0_31

LL_EXTI_ClearFallingFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearFallingFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Falling Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- This bit is set when the falling edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- FPR1 FPIFx LL_EXTI_ClearFallingFlag_0_31

LL_EXTI_IsActiveRisingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Check if the ExtLine Rising Flag is set or not for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_IsActiveRisingFlag_0_31

LL_EXTI_ReadRisingFlag_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Read ExtLine Combination Rising Flag for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **@note:** This bit is set when the selected edge event arrives on the interrupt

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_ReadRisingFlag_0_31

LL_EXTI_ClearRisingFlag_0_31

Function name

__STATIC_INLINE void LL_EXTI_ClearRisingFlag_0_31 (uint32_t ExtiLine)

Function description

Clear ExtLine Rising Flags for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)

Return values

- **None:**

Notes

- This bit is set when the Rising edge event arrives on the interrupt line. This bit is cleared by writing a 1 to the bit.
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- RPR1 RPIFx LL_EXTI_ClearRisingFlag_0_31

LL_EXTI_SetEXTISource

Function name

__STATIC_INLINE void LL_EXTI_SetEXTISource (uint32_t Port, uint32_t Line)

Function description

Configure source input for the EXTI external interrupt.

Parameters

- **Port:** This parameter can be one of the following values:
  – LL_EXTI_EXTI_PORTA
  – LL_EXTI_EXTI_PORTB
  – LL_EXTI_EXTI_PORTC
  – LL_EXTI_EXTI_PORTD
  – LL_EXTI_EXTI_PORTE
  – LL_EXTI_EXTI_PORTF (*)
  – LL_EXTI_EXTI_PORTG
  – LL_EXTI_EXTI_PORTH
  – LL_EXTI_EXTI_PORTI (*)
  – LL_EXTI_EXTI_PORTJ (*)
  (*) value not defined in all devices
- **Line:** This parameter can be one of the following values:
  – LL_EXTI_EXTI_LINE0
  – LL_EXTI_EXTI_LINE1
  – LL_EXTI_EXTI_LINE2
  – LL_EXTI_EXTI_LINE3
  – LL_EXTI_EXTI_LINE4
  – LL_EXTI_EXTI_LINE5
  – LL_EXTI_EXTI_LINE6
  – LL_EXTI_EXTI_LINE7
  – LL_EXTI_EXTI_LINE8
  – LL_EXTI_EXTI_LINE9
  – LL_EXTI_EXTI_LINE10
  – LL_EXTI_EXTI_LINE11
  – LL_EXTI_EXTI_LINE12
  – LL_EXTI_EXTI_LINE13
  – LL_EXTI_EXTI_LINE14
  – LL_EXTI_EXTI_LINE15

Return values

- **None:**

Reference Manual to LL API cross reference:

- EXTI_EXTICR1 EXTI0 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI1 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI2 LL_EXTI_SetEXTISource
- EXTI_EXTICR1 EXTI3 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI4 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI5 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI6 LL_EXTI_SetEXTISource
- EXTI_EXTICR2 EXTI7 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI8 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI9 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI10 LL_EXTI_SetEXTISource
- EXTI_EXTICR3 EXTI11 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI12 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI13 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI14 LL_EXTI_SetEXTISource
- EXTI_EXTICR4 EXTI15 LL_EXTI_SetEXTISource

LL_EXTI_GetEXTISource

Function name

__STATIC_INLINE uint32_t LL_EXTI_GetEXTISource (uint32_t Line)

Function description

Get the configured defined for specific EXTI Line.

Parameters

- **Line:** This parameter can be one of the following values:
  - LL_EXTI_EXTI_LINE0
  - LL_EXTI_EXTI_LINE1
  - LL_EXTI_EXTI_LINE2
  - LL_EXTI_EXTI_LINE3
  - LL_EXTI_EXTI_LINE4
  - LL_EXTI_EXTI_LINE5
  - LL_EXTI_EXTI_LINE6
  - LL_EXTI_EXTI_LINE7
  - LL_EXTI_EXTI_LINE8
  - LL_EXTI_EXTI_LINE9
  - LL_EXTI_EXTI_LINE10
  - LL_EXTI_EXTI_LINE11
  - LL_EXTI_EXTI_LINE12
  - LL_EXTI_EXTI_LINE13
  - LL_EXTI_EXTI_LINE14
  - LL_EXTI_EXTI_LINE15

Return values

- **Returned:** value can be one of the following values:
  - LL_EXTI_EXTI_PORTA
  - LL_EXTI_EXTI_PORTB
  - LL_EXTI_EXTI_PORTC
  - LL_EXTI_EXTI_PORTD
  - LL_EXTI_EXTI_PORTE
  - LL_EXTI_EXTI_PORTF (*)
  - LL_EXTI_EXTI_PORTG
  - LL_EXTI_EXTI_PORTH
  - LL_EXTI_EXTI_PORTI (*)
  - LL_EXTI_EXTI_PORTJ (*)

  (*) value not defined in all devices

Reference Manual to LL API cross reference:

- EXTI_EXTICR1 EXTI0 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI1 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI2 LL_EXTI_GetEXTISource
- EXTI_EXTICR1 EXTI3 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI4 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI5 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI6 LL_EXTI_GetEXTISource
- EXTI_EXTICR2 EXTI7 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI8 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI9 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI10 LL_EXTI_GetEXTISource
- EXTI_EXTICR3 EXTI11 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI12 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI13 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI14 LL_EXTI_GetEXTISource
- EXTI_EXTICR4 EXTI15 LL_EXTI_GetEXTISource

LL_EXTI_EnableSecure_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnableSecure_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Secure attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_EnableSecure_0_31

LL_EXTI_DisableSecure_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisableSecure_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Secure attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_0
  - LL_EXTI_LINE_1
  - LL_EXTI_LINE_2
  - LL_EXTI_LINE_3
  - LL_EXTI_LINE_4
  - LL_EXTI_LINE_5
  - LL_EXTI_LINE_6
  - LL_EXTI_LINE_7
  - LL_EXTI_LINE_8
  - LL_EXTI_LINE_9
  - LL_EXTI_LINE_10
  - LL_EXTI_LINE_11
  - LL_EXTI_LINE_12
  - LL_EXTI_LINE_13
  - LL_EXTI_LINE_14
  - LL_EXTI_LINE_15
  - LL_EXTI_LINE_16
  - LL_EXTI_LINE_17
  - LL_EXTI_LINE_18
  - LL_EXTI_LINE_19
  - LL_EXTI_LINE_20
  - LL_EXTI_LINE_21
  - LL_EXTI_LINE_22
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_24 (*)
  - LL_EXTI_LINE_25 (*)
  - LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_DisableSecure_0_31

LL_EXTI_IsEnabledSecure_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledSecure_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Secure attribute is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)
  – LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- SECCFGR1 SECx LL_EXTI_IsEnabledSecure_0_31

LL_EXTI_EnablePrivilege_0_31

Function name

__STATIC_INLINE void LL_EXTI_EnablePrivilege_0_31 (uint32_t ExtiLine)

Function description

Enable ExtiLine Privilege attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)
  – LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_EnablePrivilege_0_31

LL_EXTI_DisablePrivilege_0_31

Function name

__STATIC_INLINE void LL_EXTI_DisablePrivilege_0_31 (uint32_t ExtiLine)

Function description

Disable ExtiLine Privilege attribute for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)
  – LL_EXTI_LINE_ALL_0_31

Return values

- **None:**

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_DisablePrivilege_0_31

LL_EXTI_IsEnabledPrivilege_0_31

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledPrivilege_0_31 (uint32_t ExtiLine)

Function description

Indicate if ExtiLine Privilege attribute is enabled for Lines in range 0 to 31.

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  – LL_EXTI_LINE_0
  – LL_EXTI_LINE_1
  – LL_EXTI_LINE_2
  – LL_EXTI_LINE_3
  – LL_EXTI_LINE_4
  – LL_EXTI_LINE_5
  – LL_EXTI_LINE_6
  – LL_EXTI_LINE_7
  – LL_EXTI_LINE_8
  – LL_EXTI_LINE_9
  – LL_EXTI_LINE_10
  – LL_EXTI_LINE_11
  – LL_EXTI_LINE_12
  – LL_EXTI_LINE_13
  – LL_EXTI_LINE_14
  – LL_EXTI_LINE_15
  – LL_EXTI_LINE_16
  – LL_EXTI_LINE_17
  – LL_EXTI_LINE_18
  – LL_EXTI_LINE_19
  – LL_EXTI_LINE_20
  – LL_EXTI_LINE_21
  – LL_EXTI_LINE_22
  – LL_EXTI_LINE_23
  – LL_EXTI_LINE_24 (*)
  – LL_EXTI_LINE_25 (*)
  – LL_EXTI_LINE_ALL_0_31

Return values

- **State:** of bit (1 or 0).

Notes

- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- PRIVCFGR1 PRIVx LL_EXTI_IsEnabledPrivilege_0_31

LL_EXTI_LockAttributes

Function name

__STATIC_INLINE void LL_EXTI_LockAttributes (void)

Function description

Lock the secure and privilege configuration registers.

Return values

- **None:**

Reference Manual to LL API cross reference:

- LOCKR LOCK LL_EXTI_LockAttributes

LL_EXTI_GetLockAttributes

Function name

__STATIC_INLINE uint32_t LL_EXTI_GetLockAttributes (void)

Function description

Return the secure and privilege configuration registers LOCK status.

Return values

- **1:** if the secure and privilege configuration registers have been locked else 0.

Reference Manual to LL API cross reference:

- LOCKR LOCK LL_EXTI_GetLockAttributes

LL_EXTI_Init

Function name

ErrorStatus LL_EXTI_Init (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Initialize the EXTI registers according to the specified parameters in EXTI_InitStruct.

Parameters

- **EXTI_InitStruct:** pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: EXTI registers are initialized
  – ERROR: not applicable

LL_EXTI_DeInit

Function name

ErrorStatus LL_EXTI_DeInit (void)

Function description

De-initialize the EXTI registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: EXTI registers are de-initialized
  – ERROR: not applicable

LL_EXTI_StructInit

Function name

void LL_EXTI_StructInit (LL_EXTI_InitTypeDef * EXTI_InitStruct)

Function description

Set each LL_EXTI_InitTypeDef field to default value.

Parameters

- **EXTI_InitStruct:** Pointer to a LL_EXTI_InitTypeDef structure.

Return values

- **None:**

102.3 EXTI Firmware driver defines

The following section lists the various define and macros of the module.

102.3.1 EXTI

EXTI

LINE

LL_EXTI_LINE_0

Extended line 0

LL_EXTI_LINE_1

Extended line 1

LL_EXTI_LINE_2

Extended line 2

LL_EXTI_LINE_3

Extended line 3

LL_EXTI_LINE_4

Extended line 4

LL_EXTI_LINE_5

Extended line 5

LL_EXTI_LINE_6

Extended line 6

LL_EXTI_LINE_7

Extended line 7

LL_EXTI_LINE_8

Extended line 8

LL_EXTI_LINE_9

Extended line 9

LL_EXTI_LINE_10

Extended line 10

LL_EXTI_LINE_11

Extended line 11

LL_EXTI_LINE_12

Extended line 12

LL_EXTI_LINE_13

Extended line 13

LL_EXTI_LINE_14

Extended line 14

LL_EXTI_LINE_15

Extended line 15

LL_EXTI_LINE_16

Extended line 16

LL_EXTI_LINE_17
Extended line 17

LL_EXTI_LINE_18
Extended line 18

LL_EXTI_LINE_19
Extended line 19

LL_EXTI_LINE_20
Extended line 20

LL_EXTI_LINE_21
Extended line 21

LL_EXTI_LINE_22
Extended line 22

LL_EXTI_LINE_23
Extended line 23

LL_EXTI_LINE_24
Extended line 24

LL_EXTI_LINE_25
Extended line 25

LL_EXTI_LINE_ALL_0_31
ALL Extended line

LL_EXTI_LINE_NONE
None Extended line

Mode

LL_EXTI_MODE_IT
Interrupt Mode

LL_EXTI_MODE_EVENT
Event Mode

LL_EXTI_MODE_IT_EVENT
Interrupt & Event Mode

Edge Trigger

LL_EXTI_TRIGGER_NONE
No Trigger Mode

LL_EXTI_TRIGGER_RISING
Trigger Rising Mode

LL_EXTI_TRIGGER_FALLING
Trigger Falling Mode

LL_EXTI_TRIGGER_RISING_FALLING
Trigger Rising & Falling Mode

Common Write and read registers Macros

LL_EXTI_WriteReg

Description:

- Write a value in EXTI register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_EXTI_ReadReg

Description:

- Read a value in EXTI register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

103 LL FMAC Generic Driver

103.1 FMAC Firmware driver API description

The following section lists the various functions of the FMAC library.

103.1.1 Detailed description of functions

LL_FMAC_SetX1FullWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetX1FullWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure X1 full watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_SetX1FullWatermark

LL_FMAC_GetX1FullWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetX1FullWatermark (const FMAC_TypeDef * FMACx)

Function description

Return X1 full watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_GetX1FullWatermark

LL_FMAC_SetX1BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX1BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X1 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_SetX1BufferSize

LL_FMAC_GetX1BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1BufferSize (const FMAC_TypeDef * FMACx)

Function description

Return X1 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the input buffer (including the optional "headroom") (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BUF_SIZE LL_FMAC_GetX1BufferSize

LL_FMAC_SetX1Base

Function name

__STATIC_INLINE void LL_FMAC_SetX1Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X1 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_SetX1Base

LL_FMAC_GetX1Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX1Base (const FMAC_TypeDef * FMACx)

Function description

Return X1 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the input buffer (X1) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X1BUFCFG X1_BASE LL_FMAC_GetX1Base

LL_FMAC_SetX2BufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetX2BufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure X2 buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_SetX2BufferSize

LL_FMAC_GetX2BufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2BufferSize (const FMAC_TypeDef * FMACx)

Function description

Return X2 buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the coefficient buffer (value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BUF_SIZE LL_FMAC_GetX2BufferSize

LL_FMAC_SetX2Base

Function name

__STATIC_INLINE void LL_FMAC_SetX2Base (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure X2 base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_SetX2Base

LL_FMAC_GetX2Base

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetX2Base (const FMAC_TypeDef * FMACx)

Function description

Return X2 base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the coefficient buffer (X2) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_GetX2Base

LL_FMAC_SetYEmptyWatermark

Function name

__STATIC_INLINE void LL_FMAC_SetYEmptyWatermark (FMAC_TypeDef * FMACx, uint32_t Watermark)

Function description

Configure Y empty watermark.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_SetYEmptyWatermark

LL_FMAC_GetYEmptyWatermark

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetYEmptyWatermark (const FMAC_TypeDef * FMACx)

Function description

Return Y empty watermark.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_GetYEmptyWatermark

LL_FMAC_SetYBufferSize

Function name

__STATIC_INLINE void LL_FMAC_SetYBufferSize (FMAC_TypeDef * FMACx, uint8_t BufferSize)

Function description

Configure Y buffer size.

Parameters

- **FMACx:** FMAC instance
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_SetYBufferSize

LL_FMAC_GetYBufferSize

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBufferSize (const FMAC_TypeDef * FMACx)

Function description

Return Y buffer size.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Number of 16-bit words allocated to the output buffer (including the optional "headroom" - value between Min_Data=0x01 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BUF_SIZE LL_FMAC_GetYBufferSize

LL_FMAC_SetYBase

Function name

__STATIC_INLINE void LL_FMAC_SetYBase (FMAC_TypeDef * FMACx, uint8_t Base)

Function description

Configure Y base.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_SetYBase

LL_FMAC_GetYBase

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetYBase (const FMAC_TypeDef * FMACx)

Function description

Return Y base.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Base address of the output buffer (Y) within the internal memory (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- YBUFCFG Y_BASE LL_FMAC_GetYBase

LL_FMAC_EnableStart

Function name

__STATIC_INLINE void LL_FMAC_EnableStart (FMAC_TypeDef * FMACx)

Function description

Start FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_EnableStart

LL_FMAC_DisableStart

Function name

__STATIC_INLINE void LL_FMAC_DisableStart (FMAC_TypeDef * FMACx)

Function description

Stop FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_DisableStart

LL_FMAC_IsEnabledStart

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledStart (const FMAC_TypeDef * FMACx)

Function description

Check the state of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_IsEnabledStart

LL_FMAC_SetFunction

Function name

__STATIC_INLINE void LL_FMAC_SetFunction (FMAC_TypeDef * FMACx, uint32_t Function)

Function description

Configure function.

Parameters

- **FMACx:** FMAC instance
- **Function:** This parameter can be one of the following values:
  – LL_FMAC_FUNC_LOAD_X1
  – LL_FMAC_FUNC_LOAD_X2
  – LL_FMAC_FUNC_LOAD_Y
  – LL_FMAC_FUNC_CONVO_FIR
  – LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_SetFunction

LL_FMAC_GetFunction

Function name

__STATIC_INLINE uint32_t LL_FMAC_GetFunction (const FMAC_TypeDef * FMACx)

Function description

Return function.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** Returned value can be one of the following values:
  – LL_FMAC_FUNC_LOAD_X1
  – LL_FMAC_FUNC_LOAD_X2
  – LL_FMAC_FUNC_LOAD_Y
  – LL_FMAC_FUNC_CONVO_FIR
  – LL_FMAC_FUNC_IIR_DIRECT_FORM_1

Reference Manual to LL API cross reference:

- PARAM FUNC LL_FMAC_GetFunction

LL_FMAC_SetParamR

Function name

__STATIC_INLINE void LL_FMAC_SetParamR (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter R.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_SetParamR

LL_FMAC_GetParamR

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamR (const FMAC_TypeDef * FMACx)

Function description

Return input parameter R.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter R (gain, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM R LL_FMAC_GetParamR

LL_FMAC_SetParamQ

Function name

__STATIC_INLINE void LL_FMAC_SetParamQ (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter Q.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_SetParamQ

LL_FMAC_GetParamQ

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamQ (const FMAC_TypeDef * FMACx)

Function description

Return input parameter Q.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter Q (vector length, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM Q LL_FMAC_GetParamQ

LL_FMAC_SetParamP

Function name

__STATIC_INLINE void LL_FMAC_SetParamP (FMAC_TypeDef * FMACx, uint8_t Param)

Function description

Configure input parameter P.

Parameters

- **FMACx:** FMAC instance
- **Param:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_SetParamP

LL_FMAC_GetParamP

Function name

__STATIC_INLINE uint8_t LL_FMAC_GetParamP (const FMAC_TypeDef * FMACx)

Function description

Return input parameter P.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint8_t:** Parameter P (vector length, number of filter taps, etc.) (value between Min_Data=0x00 and Max_Data=0xFF).

Reference Manual to LL API cross reference:

- PARAM P LL_FMAC_GetParamP

LL_FMAC_EnableReset

Function name

__STATIC_INLINE void LL_FMAC_EnableReset (FMAC_TypeDef * FMACx)

Function description

Start the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_EnableReset

LL_FMAC_IsEnabledReset

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledReset (const FMAC_TypeDef * FMACx)

Function description

Check the state of the FMAC reset.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RESET LL_FMAC_IsEnabledReset

LL_FMAC_EnableClipping

Function name

__STATIC_INLINE void LL_FMAC_EnableClipping (FMAC_TypeDef * FMACx)

Function description

Enable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_EnableClipping

LL_FMAC_DisableClipping

Function name

__STATIC_INLINE void LL_FMAC_DisableClipping (FMAC_TypeDef * FMACx)

Function description

Disable Clipping.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_DisableClipping

LL_FMAC_IsEnabledClipping

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledClipping (const FMAC_TypeDef * FMACx)

Function description

Check Clipping State.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CLIPEN LL_FMAC_IsEnabledClipping

LL_FMAC_EnableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_EnableDMAReq_WRITE

LL_FMAC_DisableDMAReq_WRITE

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_WRITE (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA write channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_DisableDMAReq_WRITE

LL_FMAC_IsEnabledDMAReq_WRITE

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_WRITE (const FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA write channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAWEN LL_FMAC_IsEnabledDMAReq_WRITE

LL_FMAC_EnableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_EnableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Enable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_EnableDMAReq_READ

LL_FMAC_DisableDMAReq_READ

Function name

__STATIC_INLINE void LL_FMAC_DisableDMAReq_READ (FMAC_TypeDef * FMACx)

Function description

Disable FMAC DMA read channel request.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_DisableDMAReq_READ

LL_FMAC_IsEnabledDMAReq_READ

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledDMAReq_READ (const FMAC_TypeDef * FMACx)

Function description

Check FMAC DMA read channel request state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR DMAREN LL_FMAC_IsEnabledDMAReq_READ

LL_FMAC_EnableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Enable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_EnableIT_SAT

LL_FMAC_DisableIT_SAT

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_SAT (FMAC_TypeDef * FMACx)

Function description

Disable FMAC saturation error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_DisableIT_SAT

LL_FMAC_IsEnabledIT_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_SAT (const FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SATIEN LL_FMAC_IsEnabledIT_SAT

LL_FMAC_EnableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_EnableIT_UNFL

LL_FMAC_DisableIT_UNFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_UNFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC underflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_DisableIT_UNFL

LL_FMAC_IsEnabledIT_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_UNFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR UNFLIEN LL_FMAC_IsEnabledIT_UNFL

LL_FMAC_EnableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Enable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_EnableIT_OVFL

LL_FMAC_DisableIT_OVFL

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_OVFL (FMAC_TypeDef * FMACx)

Function description

Disable FMAC overflow error interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_DisableIT_OVFL

LL_FMAC_IsEnabledIT_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_OVFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OVFLIEN LL_FMAC_IsEnabledIT_OVFL

LL_FMAC_EnableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_WR (FMAC_TypeDef * FMACx)

Function description

Enable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_EnableIT_WR

LL_FMAC_DisableIT_WR

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_WR (FMAC_TypeDef * FMACx)

Function description

Disable FMAC write interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_DisableIT_WR

LL_FMAC_IsEnabledIT_WR

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_WR (const FMAC_TypeDef * FMACx)

Function description

Check FMAC write interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WIEN LL_FMAC_IsEnabledIT_WR

LL_FMAC_EnableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_EnableIT_RD (FMAC_TypeDef * FMACx)

Function description

Enable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_EnableIT_RD

LL_FMAC_DisableIT_RD

Function name

__STATIC_INLINE void LL_FMAC_DisableIT_RD (FMAC_TypeDef * FMACx)

Function description

Disable FMAC read interrupt.

Parameters

- **FMACx:** FMAC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_DisableIT_RD

LL_FMAC_IsEnabledIT_RD

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsEnabledIT_RD (const FMAC_TypeDef * FMACx)

Function description

Check FMAC read interrupt state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RIEN LL_FMAC_IsEnabledIT_RD

LL_FMAC_IsActiveFlag_SAT

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_SAT (const FMAC_TypeDef * FMACx)

Function description

Check FMAC saturation error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SAT LL_FMAC_IsActiveFlag_SAT

LL_FMAC_IsActiveFlag_UNFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_UNFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC underflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UNFL LL_FMAC_IsActiveFlag_UNFL

LL_FMAC_IsActiveFlag_OVFL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_OVFL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC overflow error flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVFL LL_FMAC_IsActiveFlag_OVFL

LL_FMAC_IsActiveFlag_X1FULL

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_X1FULL (const FMAC_TypeDef * FMACx)

Function description

Check FMAC X1 buffer full flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR X1FULL LL_FMAC_IsActiveFlag_X1FULL

LL_FMAC_IsActiveFlag_YEMPTY

Function name

__STATIC_INLINE uint32_t LL_FMAC_IsActiveFlag_YEMPTY (const FMAC_TypeDef * FMACx)

Function description

Check FMAC Y buffer empty flag state.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint32_t:** State of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR YEMPTY LL_FMAC_IsActiveFlag_YEMPTY

LL_FMAC_WriteData

Function name

__STATIC_INLINE void LL_FMAC_WriteData (FMAC_TypeDef * FMACx, uint16_t InData)

Function description

Write 16-bit input data for the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **InData:** 16-bit value to be provided as input data for FMAC processing. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WDATA WDATA LL_FMAC_WriteData

LL_FMAC_ReadData

Function name

__STATIC_INLINE uint16_t LL_FMAC_ReadData (const FMAC_TypeDef * FMACx)

Function description

Return 16-bit output data of FMAC processing.

Parameters

- **FMACx:** FMAC instance

Return values

- **uint16_t:** 16-bit output data of FMAC processing (value between Min_Data=0x0000 and Max_Data=0xFFFF).

Reference Manual to LL API cross reference:

- RDATA RDATA LL_FMAC_ReadData

LL_FMAC_ConfigX1

Function name

__STATIC_INLINE void LL_FMAC_ConfigX1 (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X1 buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the input buffer (X1) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the input buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X1BUFCFG FULL_WM LL_FMAC_ConfigX1
- X1BUFCFG X1_BASE LL_FMAC_ConfigX1
- X1BUFCFG X1_BUF_SIZE LL_FMAC_ConfigX1

LL_FMAC_ConfigX2

Function name

__STATIC_INLINE void LL_FMAC_ConfigX2 (FMAC_TypeDef * FMACx, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for X2 buffer.

Parameters

- **FMACx:** FMAC instance
- **Base:** Base address of the coefficient buffer (X2) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the coefficient buffer. This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- X2BUFCFG X2_BASE LL_FMAC_ConfigX2
- X2BUFCFG X2_BUF_SIZE LL_FMAC_ConfigX2

LL_FMAC_ConfigY

Function name

__STATIC_INLINE void LL_FMAC_ConfigY (FMAC_TypeDef * FMACx, uint32_t Watermark, uint8_t Base, uint8_t BufferSize)

Function description

Configure memory for Y buffer.

Parameters

- **FMACx:** FMAC instance
- **Watermark:** This parameter can be one of the following values:
  - LL_FMAC_WM_0_THRESHOLD_1
  - LL_FMAC_WM_1_THRESHOLD_2
  - LL_FMAC_WM_2_THRESHOLD_4
  - LL_FMAC_WM_3_THRESHOLD_8
- **Base:** Base address of the output buffer (Y) within the internal memory. This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **BufferSize:** Number of 16-bit words allocated to the output buffer (including the optional "headroom"). This parameter must be a number between Min_Data=0x01 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- YBUFCFG EMPTY_WM LL_FMAC_ConfigY
- YBUFCFG Y_BASE LL_FMAC_ConfigY
- YBUFCFG Y_BUF_SIZE LL_FMAC_ConfigY

LL_FMAC_ConfigFunc

Function name

__STATIC_INLINE void LL_FMAC_ConfigFunc (FMAC_TypeDef * FMACx, uint8_t Start, uint32_t Function, uint8_t ParamP, uint8_t ParamQ, uint8_t ParamR)

Function description

Configure the FMAC processing.

Parameters

- **FMACx:** FMAC instance
- **Start:** This parameter can be one of the following values:
  – LL_FMAC_PROCESSING_STOP
  – LL_FMAC_PROCESSING_START
- **Function:** This parameter can be one of the following values:
  – LL_FMAC_FUNC_LOAD_X1
  – LL_FMAC_FUNC_LOAD_X2
  – LL_FMAC_FUNC_LOAD_Y
  – LL_FMAC_FUNC_CONVO_FIR
  – LL_FMAC_FUNC_IIR_DIRECT_FORM_1
- **ParamP:** Parameter P (vector length, number of filter taps, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamQ:** Parameter Q (vector length, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.
- **ParamR:** Parameter R (gain, etc.). This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PARAM START LL_FMAC_ConfigFunc
- PARAM FUNC LL_FMAC_ConfigFunc
- PARAM P LL_FMAC_ConfigFunc
- PARAM Q LL_FMAC_ConfigFunc
- PARAM R LL_FMAC_ConfigFunc

LL_FMAC_Init

Function name

ErrorStatus LL_FMAC_Init (FMAC_TypeDef * FMACx)

Function description

Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **ErrorStatus:** enumeration value:
  – SUCCESS: FMAC registers are initialized
  – ERROR: FMAC registers are not initialized

LL_FMAC_DeInit

Function name

ErrorStatus LL_FMAC_DeInit (const FMAC_TypeDef * FMACx)

Function description

De-Initialize FMAC peripheral registers to their default reset values.

Parameters

- **FMACx:** FMAC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: FMAC registers are de-initialized
  - ERROR: FMAC registers are not de-initialized

103.2 FMAC Firmware driver defines

The following section lists the various define and macros of the module.

103.2.1 FMAC

FMAC

FMAC functions

LL_FMAC_FUNC_LOAD_X1

Load X1 buffer

LL_FMAC_FUNC_LOAD_X2

Load X2 buffer

LL_FMAC_FUNC_LOAD_Y

Load Y buffer

LL_FMAC_FUNC_CONVO_FIR

Convolution (FIR filter)

LL_FMAC_FUNC_IIR_DIRECT_FORM_1

IIR filter (direct form 1)

Get Flag Defines

LL_FMAC_SR_SAT

Saturation Error Flag (this helps in debugging a filter)

LL_FMAC_SR_UNFL

Underflow Error Flag

LL_FMAC_SR_OVFL

Overflow Error Flag

LL_FMAC_SR_X1FULL

X1 Buffer Full Flag

LL_FMAC_SR_YEMPTY

Y Buffer Empty Flag

IT Defines

LL_FMAC_CR_SATIEN

Saturation Error Interrupt Enable (this helps in debugging a filter)

LL_FMAC_CR_UNFLIEN

Underflow Error Interrupt Enable

LL_FMAC_CR_OVFLIEN

Overflow Error Interrupt Enable

LL_FMAC_CR_WIEN

Write Interrupt Enable

LL_FMAC_CR_RIEN

Read Interrupt Enable

FMAC processing

LL_FMAC_PROCESSING_STOP

Stop FMAC Processing

LL_FMAC_PROCESSING_START

Start FMAC Processing

FMAC watermarks

LL_FMAC_WM_0_THRESHOLD_1

Buffer full/empty flag set if there is less than 1 free/unread space.

LL_FMAC_WM_1_THRESHOLD_2

Buffer full/empty flag set if there are less than 2 free/unread spaces.

LL_FMAC_WM_2_THRESHOLD_4

Buffer full/empty flag set if there are less than 4 free/unread spaces.

LL_FMAC_WM_3_THRESHOLD_8

Buffer full/empty flag set if there are less than 8 free/empty spaces.

Common Write and read registers Macros

LL_FMAC_WriteReg

Description:
- Write a value in FMAC register.

Parameters:
- __INSTANCE__: FMAC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_FMAC_ReadReg

Description:
- Read a value in FMAC register.

Parameters:
- __INSTANCE__: FMAC Instance
- __REG__: Register to be read

Return value:
- Register: value

104 LL GPIO Generic Driver

104.1 GPIO Firmware driver registers structures

104.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32u5xx_ll_gpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***
- ***uint32_t Speed***
- ***uint32_t OutputType***
- ***uint32_t Pull***
- ***uint32_t Alternate***

Field Documentation

- ***uint32_t LL_GPIO_InitTypeDef::Pin***
  Specifies the GPIO pins to be configured. This parameter can be any value of ***GPIO_LL_EC_PIN***
- ***uint32_t LL_GPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***GPIO_LL_EC_MODE***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinMode()**.
- ***uint32_t LL_GPIO_InitTypeDef::Speed***
  Specifies the speed for the selected pins. This parameter can be a value of ***GPIO_LL_EC_SPEED***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinSpeed()**.
- ***uint32_t LL_GPIO_InitTypeDef::OutputType***
  Specifies the operating output type for the selected pins. This parameter can be a value of ***GPIO_LL_EC_OUTPUT***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinOutputType()**.
- ***uint32_t LL_GPIO_InitTypeDef::Pull***
  Specifies the operating Pull-up/Pull down for the selected pins. This parameter can be a value of ***GPIO_LL_EC_PULL***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetPinPull()**.
- ***uint32_t LL_GPIO_InitTypeDef::Alternate***
  Specifies the Peripheral to be connected to the selected pins. This parameter can be a value of ***GPIO_LL_EC_AF***.GPIO HW configuration can be modified afterwards using unitary function **LL_GPIO_SetAFPin_0_7()** and **LL_GPIO_SetAFPin_8_15()**.

104.2 GPIO Firmware driver API description

The following section lists the various functions of the GPIO library.

104.2.1 Detailed description of functions

LL_GPIO_SetPinMode

Function name

__STATIC_INLINE void LL_GPIO_SetPinMode (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Mode)

Function description

Configure gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  – LL_GPIO_MODE_INPUT
  – LL_GPIO_MODE_OUTPUT
  – LL_GPIO_MODE_ALTERNATE
  – LL_GPIO_MODE_ANALOG

Return values

- **None:**

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_SetPinMode

LL_GPIO_GetPinMode

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinMode (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio mode for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_MODE_INPUT
  - LL_GPIO_MODE_OUTPUT
  - LL_GPIO_MODE_ALTERNATE
  - LL_GPIO_MODE_ANALOG

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_GPIO_GetPinMode

LL_GPIO_SetPinOutputType

Function name

__STATIC_INLINE void LL_GPIO_SetPinOutputType (GPIO_TypeDef * GPIOx, uint32_t PinMask, uint32_t OutputType)

Function description

Configure gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
  – LL_GPIO_PIN_ALL
- **OutputType:** This parameter can be one of the following values:
  – LL_GPIO_OUTPUT_PUSHPULL
  – LL_GPIO_OUTPUT_OPENDRAIN

Return values

- **None:**

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_SetPinOutputType

LL_GPIO_GetPinOutputType

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinOutputType (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio output type for several pins on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_OUTPUT_PUSHPULL
  - LL_GPIO_OUTPUT_OPENDRAIN

Notes

- Output type as to be set when gpio pin is in output or alternate modes. Possible type are Push-pull or Open-drain.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- OTYPER OTy LL_GPIO_GetPinOutputType

LL_GPIO_SetPinSpeed

Function name

__STATIC_INLINE void LL_GPIO_SetPinSpeed (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Speed)

Function description

Configure gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Speed:** This parameter can be one of the following values:
  - LL_GPIO_SPEED_FREQ_LOW
  - LL_GPIO_SPEED_FREQ_MEDIUM
  - LL_GPIO_SPEED_FREQ_HIGH
  - LL_GPIO_SPEED_FREQ_VERY_HIGH

Return values

- **None:**

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_SetPinSpeed

LL_GPIO_GetPinSpeed

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinSpeed (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio speed for a dedicated pin on dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  – LL_GPIO_SPEED_FREQ_LOW
  – LL_GPIO_SPEED_FREQ_MEDIUM
  – LL_GPIO_SPEED_FREQ_HIGH
  – LL_GPIO_SPEED_FREQ_VERY_HIGH

Notes

- I/O speed can be Low, Medium, Fast or High speed.
- Warning: only one pin can be passed as parameter.
- Refer to datasheet for frequency specifications and the power supply and load conditions for each speed.

Reference Manual to LL API cross reference:

- OSPEEDR OSPEEDy LL_GPIO_GetPinSpeed

LL_GPIO_SetPinPull

Function name

__STATIC_INLINE void LL_GPIO_SetPinPull (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Pull)

Function description

Configure gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
- **Pull:** This parameter can be one of the following values:
  – LL_GPIO_PULL_NO
  – LL_GPIO_PULL_UP
  – LL_GPIO_PULL_DOWN

Return values

- **None:**

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_SetPinPull

LL_GPIO_GetPinPull

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetPinPull (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio pull-up or pull-down for a dedicated pin on a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_PULL_NO
  - LL_GPIO_PULL_UP
  - LL_GPIO_PULL_DOWN

Notes

- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- PUPDR PUPDy LL_GPIO_GetPinPull

LL_GPIO_SetAFPin_0_7

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_0_7 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_SetAFPin_0_7

LL_GPIO_GetAFPin_0_7

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_0_7 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 0 to 7 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7

Return values

- **Returned:** value can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Reference Manual to LL API cross reference:

- AFRL AFSELy LL_GPIO_GetAFPin_0_7

LL_GPIO_SetAFPin_8_15

Function name

__STATIC_INLINE void LL_GPIO_SetAFPin_8_15 (GPIO_TypeDef * GPIOx, uint32_t Pin, uint32_t Alternate)

Function description

Configure gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
- **Alternate:** This parameter can be one of the following values:
  - LL_GPIO_AF_0
  - LL_GPIO_AF_1
  - LL_GPIO_AF_2
  - LL_GPIO_AF_3
  - LL_GPIO_AF_4
  - LL_GPIO_AF_5
  - LL_GPIO_AF_6
  - LL_GPIO_AF_7
  - LL_GPIO_AF_8
  - LL_GPIO_AF_9
  - LL_GPIO_AF_10
  - LL_GPIO_AF_11
  - LL_GPIO_AF_12
  - LL_GPIO_AF_13
  - LL_GPIO_AF_14
  - LL_GPIO_AF_15

Return values

- **None:**

Notes

- Possible values are from AF0 to AF15 depending on target.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_SetAFPin_8_15

LL_GPIO_GetAFPin_8_15

Function name

__STATIC_INLINE uint32_t LL_GPIO_GetAFPin_8_15 (const GPIO_TypeDef * GPIOx, uint32_t Pin)

Function description

Return gpio alternate function of a dedicated pin from 8 to 15 for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **Pin:** This parameter can be one of the following values:
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  – LL_GPIO_AF_0
  – LL_GPIO_AF_1
  – LL_GPIO_AF_2
  – LL_GPIO_AF_3
  – LL_GPIO_AF_4
  – LL_GPIO_AF_5
  – LL_GPIO_AF_6
  – LL_GPIO_AF_7
  – LL_GPIO_AF_8
  – LL_GPIO_AF_9
  – LL_GPIO_AF_10
  – LL_GPIO_AF_11
  – LL_GPIO_AF_12
  – LL_GPIO_AF_13
  – LL_GPIO_AF_14
  – LL_GPIO_AF_15

Notes

- Possible values are from AF0 to AF15 depending on target.

Reference Manual to LL API cross reference:

- AFRH AFSELy LL_GPIO_GetAFPin_8_15

LL_GPIO_LockPin

Function name

__STATIC_INLINE void LL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Lock configuration of several pins for a dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Notes

- When the lock sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next reset.
- Each lock bit freezes a specific configuration register (control and alternate function registers).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_LockPin

LL_GPIO_IsPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsPinLocked (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return 1 if all pins passed as parameter, of a dedicated port, are locked.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKy LL_GPIO_IsPinLocked

LL_GPIO_IsAnyPinLocked

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsAnyPinLocked (const GPIO_TypeDef * GPIOx)

Function description

Return 1 if one of the pin of a dedicated port is locked.

Parameters

- **GPIOx:** GPIO Port

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- LCKR LCKK LL_GPIO_IsAnyPinLocked

LL_GPIO_ReadInputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadInputPort (const GPIO_TypeDef * GPIOx)

Function description

Return full input data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Input:** data register value of port

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_ReadInputPort

LL_GPIO_IsInputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsInputPinSet (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IDR IDy LL_GPIO_IsInputPinSet

LL_GPIO_WriteOutputPort

Function name

__STATIC_INLINE void LL_GPIO_WriteOutputPort (GPIO_TypeDef * GPIOx, uint32_t PortValue)

Function description

Write output data register for the port.

Parameters

- **GPIOx:** GPIO Port
- **PortValue:** Level value for each pin of the port Value between 0 and 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_WriteOutputPort

LL_GPIO_ReadOutputPort

Function name

__STATIC_INLINE uint32_t LL_GPIO_ReadOutputPort (const GPIO_TypeDef * GPIOx)

Function description

Return full output data register value for a dedicated port.

Parameters

- **GPIOx:** GPIO Port

Return values

- **Output:** data register value of port

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_ReadOutputPort

LL_GPIO_IsOutputPinSet

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsOutputPinSet (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
  – LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_IsOutputPinSet

LL_GPIO_SetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_SetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BSy LL_GPIO_SetOutputPin

LL_GPIO_ResetOutputPin

Function name

__STATIC_INLINE void LL_GPIO_ResetOutputPin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Set several pins to low level on dedicated gpio port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
  – LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BRR BRy LL_GPIO_ResetOutputPin

LL_GPIO_TogglePin

Function name

__STATIC_INLINE void LL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Toggle data value for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_GPIO_TogglePin

LL_GPIO_EnableHighSPeedLowVoltage

Function name

__STATIC_INLINE void LL_GPIO_EnableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Enable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

Reference Manual to LL API cross reference:

- HSLVR HSLVy LL_GPIO_EnableHighSPeedLowVoltage

LL_GPIO_DisableHighSPeedLowVoltage

Function name

__STATIC_INLINE void LL_GPIO_DisableHighSPeedLowVoltage (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Disable speed optimization for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15

Return values

- **None:**

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

Reference Manual to LL API cross reference:

- HSLVR HSLVy LL_GPIO_DisableHighSPeedLowVoltage

LL_GPIO_IsEnabledHighSPeedLowVoltage

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsEnabledHighSPeedLowVoltage (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if speed optimization for several pin of dedicated port is enabled or not.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15

Return values

- **State:** of bit (1 or 0).

Notes

- Not all I/Os support the HSLV mode. Refer to the I/O structure in the corresponding datasheet for the list of I/Os supporting this feature. Other I/Os HSLV configuration must be kept at reset value.
- It must be used only if the I/O supply voltage is below 2.7 V.

Reference Manual to LL API cross reference:

- HSLVR HSLVy LL_GPIO_IsEnabledHighSPeedLowVoltage

LL_GPIO_EnablePinSecure

Function name

__STATIC_INLINE void LL_GPIO_EnablePinSecure (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Enable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_EnablePinSecure

LL_GPIO_DisablePinSecure

Function name

__STATIC_INLINE void LL_GPIO_DisablePinSecure (GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Disable secure write only access for several pin of dedicated port.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_GPIO_PIN_0
  – LL_GPIO_PIN_1
  – LL_GPIO_PIN_2
  – LL_GPIO_PIN_3
  – LL_GPIO_PIN_4
  – LL_GPIO_PIN_5
  – LL_GPIO_PIN_6
  – LL_GPIO_PIN_7
  – LL_GPIO_PIN_8
  – LL_GPIO_PIN_9
  – LL_GPIO_PIN_10
  – LL_GPIO_PIN_11
  – LL_GPIO_PIN_12
  – LL_GPIO_PIN_13
  – LL_GPIO_PIN_14
  – LL_GPIO_PIN_15
  – LL_GPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_DisablePinSecure

LL_GPIO_IsEnabledPinSecure

Function name

__STATIC_INLINE uint32_t LL_GPIO_IsEnabledPinSecure (const GPIO_TypeDef * GPIOx, uint32_t PinMask)

Function description

Return if secure write only access for several pin of dedicated port is enabled or not.

Parameters

- **GPIOx:** GPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_GPIO_PIN_0
  - LL_GPIO_PIN_1
  - LL_GPIO_PIN_2
  - LL_GPIO_PIN_3
  - LL_GPIO_PIN_4
  - LL_GPIO_PIN_5
  - LL_GPIO_PIN_6
  - LL_GPIO_PIN_7
  - LL_GPIO_PIN_8
  - LL_GPIO_PIN_9
  - LL_GPIO_PIN_10
  - LL_GPIO_PIN_11
  - LL_GPIO_PIN_12
  - LL_GPIO_PIN_13
  - LL_GPIO_PIN_14
  - LL_GPIO_PIN_15
  - LL_GPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SECCFGR SECy LL_GPIO_IsEnabledPinSecure

LL_GPIO_DeInit

Function name

ErrorStatus LL_GPIO_DeInit (const GPIO_TypeDef * GPIOx)

Function description

De-initialize GPIO registers (Registers restored to their default values).

Parameters

- **GPIOx:** GPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: GPIO registers are de-initialized
  - ERROR: Wrong GPIO Port

LL_GPIO_Init

Function name

ErrorStatus LL_GPIO_Init (GPIO_TypeDef * GPIOx, LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Initialize GPIO registers according to the specified parameters in GPIO_InitStruct.

Parameters

- **GPIOx:** GPIO Port
- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: GPIO registers are initialized according to GPIO_InitStruct content
  – ERROR: Not applicable

LL_GPIO_StructInit

Function name

void LL_GPIO_StructInit (LL_GPIO_InitTypeDef * GPIO_InitStruct)

Function description

Set each LL_GPIO_InitTypeDef field to default value.

Parameters

- **GPIO_InitStruct:** pointer to a LL_GPIO_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

104.3 GPIO Firmware driver defines

The following section lists the various define and macros of the module.

104.3.1 GPIO

GPIO

Alternate Function

LL_GPIO_AF_0

Select alternate function 0

LL_GPIO_AF_1

Select alternate function 1

LL_GPIO_AF_2

Select alternate function 2

LL_GPIO_AF_3

Select alternate function 3

LL_GPIO_AF_4

Select alternate function 4

LL_GPIO_AF_5

Select alternate function 5

LL_GPIO_AF_6

Select alternate function 6

LL_GPIO_AF_7

Select alternate function 7

LL_GPIO_AF_8
Select alternate function 8

LL_GPIO_AF_9
Select alternate function 9

LL_GPIO_AF_10
Select alternate function 10

LL_GPIO_AF_11
Select alternate function 11

LL_GPIO_AF_12
Select alternate function 12

LL_GPIO_AF_13
Select alternate function 13

LL_GPIO_AF_14
Select alternate function 14

LL_GPIO_AF_15
Select alternate function 15

Mode

LL_GPIO_MODE_INPUT
Select input mode

LL_GPIO_MODE_OUTPUT
Select output mode

LL_GPIO_MODE_ALTERNATE
Select alternate function mode

LL_GPIO_MODE_ANALOG
Select analog mode

Output Type

LL_GPIO_OUTPUT_PUSHPULL
Select push-pull as output type

LL_GPIO_OUTPUT_OPENDRAIN
Select open-drain as output type

PIN

LL_GPIO_PIN_0
Select pin 0

LL_GPIO_PIN_1
Select pin 1

LL_GPIO_PIN_2
Select pin 2

LL_GPIO_PIN_3

Select pin 3

LL_GPIO_PIN_4

Select pin 4

LL_GPIO_PIN_5

Select pin 5

LL_GPIO_PIN_6

Select pin 6

LL_GPIO_PIN_7

Select pin 7

LL_GPIO_PIN_8

Select pin 8

LL_GPIO_PIN_9

Select pin 9

LL_GPIO_PIN_10

Select pin 10

LL_GPIO_PIN_11

Select pin 11

LL_GPIO_PIN_12

Select pin 12

LL_GPIO_PIN_13

Select pin 13

LL_GPIO_PIN_14

Select pin 14

LL_GPIO_PIN_15

Select pin 15

LL_GPIO_PIN_ALL

Select all pins

Pull Up Pull Down

LL_GPIO_PULL_NO

Select I/O no pull

LL_GPIO_PULL_UP

Select I/O pull up

LL_GPIO_PULL_DOWN

Select I/O pull down

Output Speed

LL_GPIO_SPEED_FREQ_LOW

Select I/O low output speed

LL_GPIO_SPEED_FREQ_MEDIUM

Select I/O medium output speed

LL_GPIO_SPEED_FREQ_HIGH

Select I/O fast output speed

LL_GPIO_SPEED_FREQ_VERY_HIGH

Select I/O high output speed

Common Write and read registers Macros

LL_GPIO_WriteReg

Description:

- Write a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_GPIO_ReadReg

Description:

- Read a value in GPIO register.

Parameters:

- __INSTANCE__: GPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

GPIO Exported Constants

LL_GPIO_SPEED_LOW

LL_GPIO_SPEED_MEDIUM

LL_GPIO_SPEED_FAST

LL_GPIO_SPEED_HIGH

105 LL I2C Generic Driver

105.1 I2C Firmware driver registers structures

105.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32u5xx_ll_i2c.h

Data Fields

- ***uint32_t PeripheralMode***
- ***uint32_t Timing***
- ***uint32_t AnalogFilter***
- ***uint32_t DigitalFilter***
- ***uint32_t OwnAddress1***
- ***uint32_t TypeAcknowledge***
- ***uint32_t OwnAddrSize***

Field Documentation

- ***uint32_t LL_I2C_InitTypeDef::PeripheralMode***
  Specifies the peripheral mode. This parameter can be a value of *I2C_LL_EC_PERIPHERAL_MODE*.This feature can be modified afterwards using unitary function **LL_I2C_SetMode()**.
- ***uint32_t LL_I2C_InitTypeDef::Timing***
  Specifies the SDA setup, hold time and the SCL high, low period values. This parameter must be set by referring to the STM32CubeMX Tool and the helper macro **__LL_I2C_CONVERT_TIMINGS()**.This feature can be modified afterwards using unitary function **LL_I2C_SetTiming()**.
- ***uint32_t LL_I2C_InitTypeDef::AnalogFilter***
  Enables or disables analog noise filter. This parameter can be a value of *I2C_LL_EC_ANALOGFILTER_SELECTION*.This feature can be modified afterwards using unitary functions **LL_I2C_EnableAnalogFilter()** or **LL_I2C_DisableAnalogFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::DigitalFilter***
  Configures the digital noise filter. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0x0F.This feature can be modified afterwards using unitary function **LL_I2C_SetDigitalFilter()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddress1***
  Specifies the device own address 1. This parameter must be a value between Min_Data = 0x00 and Max_Data = 0x3FF.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.
- ***uint32_t LL_I2C_InitTypeDef::TypeAcknowledge***
  Specifies the ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte. This parameter can be a value of *I2C_LL_EC_I2C_ACKNOWLEDGE*.This feature can be modified afterwards using unitary function **LL_I2C_AcknowledgeNextData()**.
- ***uint32_t LL_I2C_InitTypeDef::OwnAddrSize***
  Specifies the device own address 1 size (7-bit or 10-bit). This parameter can be a value of *I2C_LL_EC_OWNADDRESS1*.This feature can be modified afterwards using unitary function **LL_I2C_SetOwnAddress1()**.

105.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

105.2.1 Detailed description of functions

LL_I2C_Enable

Function name

__STATIC_INLINE void LL_I2C_Enable (I2C_TypeDef * I2Cx)

Function description

Enable I2C peripheral (PE = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Enable

LL_I2C_Disable

Function name

__STATIC_INLINE void LL_I2C_Disable (I2C_TypeDef * I2Cx)

Function description

Disable I2C peripheral (PE = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When PE = 0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_Disable

LL_I2C_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabled (const I2C_TypeDef * I2Cx)

Function description

Check if the I2C peripheral is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PE LL_I2C_IsEnabled

LL_I2C_ConfigFilters

Function name

__STATIC_INLINE void LL_I2C_ConfigFilters (I2C_TypeDef * I2Cx, uint32_t AnalogFilter, uint32_t DigitalFilter)

Function description

Configure Noise Filters (Analog and Digital).

Parameters

- **I2Cx:** I2C Instance.
- **AnalogFilter:** This parameter can be one of the following values:
  - LL_I2C_ANALOGFILTER_ENABLE
  - LL_I2C_ANALOGFILTER_DISABLE
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. The filters can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_ConfigFilters
- CR1 DNF LL_I2C_ConfigFilters

LL_I2C_SetDigitalFilter

Function name

__STATIC_INLINE void LL_I2C_SetDigitalFilter (I2C_TypeDef * I2Cx, uint32_t DigitalFilter)

Function description

Configure Digital Noise Filter.

Parameters

- **I2Cx:** I2C Instance.
- **DigitalFilter:** This parameter must be a value between Min_Data=0x00 (Digital filter disabled) and Max_Data=0x0F (Digital filter enabled and filtering capability up to 15*ti2cclk). This parameter is used to configure the digital noise filter on SDA and SCL input. The digital filter will filter spikes with a length of up to DNF[3:0]*ti2cclk.

Return values

- **None:**

Notes

- If the analog filter is also enabled, the digital filter is added to analog filter. This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_SetDigitalFilter

LL_I2C_GetDigitalFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDigitalFilter (const I2C_TypeDef * I2Cx)

Function description

Get the current Digital Noise Filter configuration.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- CR1 DNF LL_I2C_GetDigitalFilter

LL_I2C_EnableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_EnableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Enable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_EnableAnalogFilter

LL_I2C_DisableAnalogFilter

Function name

__STATIC_INLINE void LL_I2C_DisableAnalogFilter (I2C_TypeDef * I2Cx)

Function description

Disable Analog Noise Filter.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This filter can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_DisableAnalogFilter

LL_I2C_IsEnabledAnalogFilter

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAnalogFilter (const I2C_TypeDef * I2Cx)

Function description

Check if Analog Noise Filter is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ANFOFF LL_I2C_IsEnabledAnalogFilter

LL_I2C_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Enable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_EnableDMAReq_TX

LL_I2C_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_TX (I2C_TypeDef * I2Cx)

Function description

Disable DMA transmission requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_DisableDMAReq_TX

LL_I2C_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_TX (const I2C_TypeDef * I2Cx)

Function description

Check if DMA transmission requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXDMAEN LL_I2C_IsEnabledDMAReq_TX

LL_I2C_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_EnableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Enable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_EnableDMAReq_RX

LL_I2C_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2C_DisableDMAReq_RX (I2C_TypeDef * I2Cx)

Function description

Disable DMA reception requests.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_DisableDMAReq_RX

LL_I2C_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledDMAReq_RX (const I2C_TypeDef * I2Cx)

Function description

Check if DMA reception requests are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXDMAEN LL_I2C_IsEnabledDMAReq_RX

LL_I2C_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_DMA_GetRegAddr (const I2C_TypeDef * I2Cx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **I2Cx:** I2C Instance
- **Direction:** This parameter can be one of the following values:
  – LL_I2C_DMA_REG_DATA_TRANSMIT
  – LL_I2C_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_DMA_GetRegAddr
- RXDR RXDATA LL_I2C_DMA_GetRegAddr

LL_I2C_EnableClockStretching

Function name

__STATIC_INLINE void LL_I2C_EnableClockStretching (I2C_TypeDef * I2Cx)

Function description

Enable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_EnableClockStretching

LL_I2C_DisableClockStretching

Function name

__STATIC_INLINE void LL_I2C_DisableClockStretching (I2C_TypeDef * I2Cx)

Function description

Disable Clock stretching.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_DisableClockStretching

LL_I2C_IsEnabledClockStretching

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledClockStretching (const I2C_TypeDef * I2Cx)

Function description

Check if Clock stretching is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NOSTRETCH LL_I2C_IsEnabledClockStretching

LL_I2C_EnableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_EnableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Enable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_EnableSlaveByteControl

LL_I2C_DisableSlaveByteControl

Function name

__STATIC_INLINE void LL_I2C_DisableSlaveByteControl (I2C_TypeDef * I2Cx)

Function description

Disable hardware byte control in slave mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_DisableSlaveByteControl

LL_I2C_IsEnabledSlaveByteControl

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSlaveByteControl (const I2C_TypeDef * I2Cx)

Function description

Check if hardware byte control in slave mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 SBC LL_I2C_IsEnabledSlaveByteControl

LL_I2C_EnableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_EnableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Enable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.
- This bit can only be programmed when Digital Filter is disabled.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_EnableWakeUpFromStop

LL_I2C_DisableWakeUpFromStop

Function name

__STATIC_INLINE void LL_I2C_DisableWakeUpFromStop (I2C_TypeDef * I2Cx)

Function description

Disable Wakeup from STOP.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_DisableWakeUpFromStop

LL_I2C_IsEnabledWakeUpFromStop

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledWakeUpFromStop (const I2C_TypeDef * I2Cx)

Function description

Check if Wakeup from STOP is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_I2C_WAKEUP_FROMSTOP_INSTANCE(I2Cx) can be used to check whether or not WakeUpFromStop feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 WUPEN LL_I2C_IsEnabledWakeUpFromStop

LL_I2C_EnableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_EnableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Enable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When enabled the Address 0x00 is ACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_EnableGeneralCall

LL_I2C_DisableGeneralCall

Function name

__STATIC_INLINE void LL_I2C_DisableGeneralCall (I2C_TypeDef * I2Cx)

Function description

Disable General Call.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- When disabled the Address 0x00 is NACKed.

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_DisableGeneralCall

LL_I2C_IsEnabledGeneralCall

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledGeneralCall (const I2C_TypeDef * I2Cx)

Function description

Check if General Call is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 GCEN LL_I2C_IsEnabledGeneralCall

LL_I2C_EnableFastModePlus

Function name

__STATIC_INLINE void LL_I2C_EnableFastModePlus (I2C_TypeDef * I2Cx)

Function description

Enable I2C Fast Mode Plus (FMP = 1).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- 20mA I/O drive enable

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_EnableFastModePlus

LL_I2C_DisableFastModePlus

Function name

__STATIC_INLINE void LL_I2C_DisableFastModePlus (I2C_TypeDef * I2Cx)

Function description

Disable I2C Fast Mode Plus (FMP = 0).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- 20mA I/O drive disable

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_DisableFastModePlus

LL_I2C_IsEnabledFastModePlus

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledFastModePlus (const I2C_TypeDef * I2Cx)

Function description

Check if the I2C Fast Mode Plus is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FMP LL_I2C_IsEnabledFastModePlus

LL_I2C_EnableAutoClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableAutoClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable automatic clear of ADDR flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_EnableAutoClearFlag_ADDR

LL_I2C_DisableAutoClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableAutoClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable automatic clear of ADDR flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_DisableAutoClearFlag_ADDR

LL_I2C_IsEnabledAutoClearFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoClearFlag_ADDR (const I2C_TypeDef * I2Cx)

Function description

Check if the automatic clear of ADDR flag is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRACLR LL_I2C_IsEnabledAutoClearFlag_ADDR

LL_I2C_EnableAutoClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableAutoClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Enable automatic clear of STOP flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_EnableAutoClearFlag_STOP

LL_I2C_DisableAutoClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableAutoClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Disable automatic clear of STOP flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_DisableAutoClearFlag_STOP

LL_I2C_IsEnabledAutoClearFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoClearFlag_STOP (const I2C_TypeDef * I2Cx)

Function description

Check if the automatic clear of STOP flag is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPFACLR LL_I2C_IsEnabledAutoClearFlag_STOP

LL_I2C_SetMasterAddressingMode

Function name

__STATIC_INLINE void LL_I2C_SetMasterAddressingMode (I2C_TypeDef * I2Cx, uint32_t AddressingMode)

Function description

Configure the Master to operate in 7-bit or 10-bit addressing mode.

Parameters

- **I2Cx:** I2C Instance.
- **AddressingMode:** This parameter can be one of the following values:
  – LL_I2C_ADDRESSING_MODE_7BIT
  – LL_I2C_ADDRESSING_MODE_10BIT

Return values

- **None:**

Notes

- Changing this bit is not allowed, when the START bit is set.

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_SetMasterAddressingMode

LL_I2C_GetMasterAddressingMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMasterAddressingMode (const I2C_TypeDef * I2Cx)

Function description

Get the Master addressing mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  – LL_I2C_ADDRESSING_MODE_7BIT
  – LL_I2C_ADDRESSING_MODE_10BIT

Reference Manual to LL API cross reference:

- CR2 ADD10 LL_I2C_GetMasterAddressingMode

LL_I2C_SetOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress1 (I2C_TypeDef * I2Cx, uint32_t OwnAddress1, uint32_t OwnAddrSize)

Function description

Set the Own Address1.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress1:** This parameter must be a value between Min_Data=0 and Max_Data=0x3FF.
- **OwnAddrSize:** This parameter can be one of the following values:
  – LL_I2C_OWNADDRESS1_7BIT
  – LL_I2C_OWNADDRESS1_10BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1 LL_I2C_SetOwnAddress1
- OAR1 OA1MODE LL_I2C_SetOwnAddress1

LL_I2C_EnableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_EnableOwnAddress1

LL_I2C_DisableOwnAddress1

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress1 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address1 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_DisableOwnAddress1

LL_I2C_IsEnabledOwnAddress1

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress1 (const I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR1 OA1EN LL_I2C_IsEnabledOwnAddress1

LL_I2C_SetOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_SetOwnAddress2 (I2C_TypeDef * I2Cx, uint32_t OwnAddress2, uint32_t OwnAddrMask)

Function description

Set the 7bits Own Address2.

Parameters

- **I2Cx:** I2C Instance.
- **OwnAddress2:** Value between Min_Data=0 and Max_Data=0x7F.
- **OwnAddrMask:** This parameter can be one of the following values:
  - LL_I2C_OWNADDRESS2_NOMASK
  - LL_I2C_OWNADDRESS2_MASK01
  - LL_I2C_OWNADDRESS2_MASK02
  - LL_I2C_OWNADDRESS2_MASK03
  - LL_I2C_OWNADDRESS2_MASK04
  - LL_I2C_OWNADDRESS2_MASK05
  - LL_I2C_OWNADDRESS2_MASK06
  - LL_I2C_OWNADDRESS2_MASK07

Return values

- **None:**

Notes

- This action has no effect if own address2 is enabled.

Reference Manual to LL API cross reference:

- OAR2 OA2 LL_I2C_SetOwnAddress2
- OAR2 OA2MSK LL_I2C_SetOwnAddress2

LL_I2C_EnableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_EnableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Enable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_EnableOwnAddress2

LL_I2C_DisableOwnAddress2

Function name

__STATIC_INLINE void LL_I2C_DisableOwnAddress2 (I2C_TypeDef * I2Cx)

Function description

Disable acknowledge on Own Address2 match address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_DisableOwnAddress2

LL_I2C_IsEnabledOwnAddress2

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledOwnAddress2 (const I2C_TypeDef * I2Cx)

Function description

Check if Own Address1 acknowledge is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- OAR2 OA2EN LL_I2C_IsEnabledOwnAddress2

LL_I2C_SetTiming

Function name

__STATIC_INLINE void LL_I2C_SetTiming (I2C_TypeDef * I2Cx, uint32_t Timing)

Function description

Configure the SDA setup, hold time and the SCL high, low period.

Parameters

- **I2Cx:** I2C Instance.
- **Timing:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFFFFFFF.

Return values

- **None:**

Notes

- This bit can only be programmed when the I2C is disabled (PE = 0).
- This parameter is computed with the STM32CubeMX Tool.

Reference Manual to LL API cross reference:

- TIMINGR TIMINGR LL_I2C_SetTiming

LL_I2C_GetTimingPrescaler

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTimingPrescaler (const I2C_TypeDef * I2Cx)

Function description

Get the Timing Prescaler setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR PRESC LL_I2C_GetTimingPrescaler

LL_I2C_GetClockLowPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockLowPeriod (const I2C_TypeDef * I2Cx)

Function description

Get the SCL low period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLL LL_I2C_GetClockLowPeriod

LL_I2C_GetClockHighPeriod

Function name

__STATIC_INLINE uint32_t LL_I2C_GetClockHighPeriod (const I2C_TypeDef * I2Cx)

Function description

Get the SCL high period setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TIMINGR SCLH LL_I2C_GetClockHighPeriod

LL_I2C_GetDataHoldTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataHoldTime (const I2C_TypeDef * I2Cx)

Function description

Get the SDA hold time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SDADEL LL_I2C_GetDataHoldTime

LL_I2C_GetDataSetupTime

Function name

__STATIC_INLINE uint32_t LL_I2C_GetDataSetupTime (const I2C_TypeDef * I2Cx)

Function description

Get the SDA setup time.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xF

Reference Manual to LL API cross reference:

- TIMINGR SCLDEL LL_I2C_GetDataSetupTime

LL_I2C_SetMode

Function name

__STATIC_INLINE void LL_I2C_SetMode (I2C_TypeDef * I2Cx, uint32_t PeripheralMode)

Function description

Configure peripheral mode.

Parameters

- **I2Cx:** I2C Instance.
- **PeripheralMode:** This parameter can be one of the following values:
  – LL_I2C_MODE_I2C
  – LL_I2C_MODE_SMBUS_HOST
  – LL_I2C_MODE_SMBUS_DEVICE
  – LL_I2C_MODE_SMBUS_DEVICE_ARP

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_SetMode
- CR1 SMBDEN LL_I2C_SetMode

LL_I2C_GetMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetMode (const I2C_TypeDef * I2Cx)

Function description

Get peripheral mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_MODE_I2C
  - LL_I2C_MODE_SMBUS_HOST
  - LL_I2C_MODE_SMBUS_DEVICE
  - LL_I2C_MODE_SMBUS_DEVICE_ARP

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 SMBHEN LL_I2C_GetMode
- CR1 SMBDEN LL_I2C_GetMode

LL_I2C_EnableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Enable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is drived low and Alert Response Address Header acknowledge is enabled. SMBus Host mode:SMBus Alert pin management is supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_EnableSMBusAlert

LL_I2C_DisableSMBusAlert

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusAlert (I2C_TypeDef * I2Cx)

Function description

Disable SMBus alert (Host or Device mode)

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- SMBus Device mode: SMBus Alert pin is not drived (can be used as a standard GPIO) and Alert Response Address Header acknowledge is disabled. SMBus Host mode:SMBus Alert pin management is not supported.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_DisableSMBusAlert

LL_I2C_IsEnabledSMBusAlert

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusAlert (const I2C_TypeDef * I2Cx)

Function description

Check if SMBus alert (Host or Device mode) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 ALERTEN LL_I2C_IsEnabledSMBusAlert

LL_I2C_EnableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Enable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_EnableSMBusPEC

LL_I2C_DisableSMBusPEC

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusPEC (I2C_TypeDef * I2Cx)

Function description

Disable SMBus Packet Error Calculation (PEC).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_DisableSMBusPEC

LL_I2C_IsEnabledSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPEC (const I2C_TypeDef * I2Cx)

Function description

Check if SMBus Packet Error Calculation (PEC) is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR1 PECEN LL_I2C_IsEnabledSMBusPEC

LL_I2C_ConfigSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_ConfigSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t TimeoutA, uint32_t TimeoutAMode, uint32_t TimeoutB)

Function description

Configure the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.
- **TimeoutAMode:** This parameter can be one of the following values:
  – LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  – LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH
- **TimeoutB:**

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This configuration can only be programmed when associated Timeout is disabled (TimeoutA and/ orTimeoutB).

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIDLE LL_I2C_ConfigSMBusTimeout
- TIMEOUTR TIMEOUTB LL_I2C_ConfigSMBusTimeout

LL_I2C_SetSMBusTimeoutA

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutA (I2C_TypeDef * I2Cx, uint32_t TimeoutA)

Function description

Configure the SMBus Clock TimeoutA (SCL low timeout or SCL and SDA high timeout depends on TimeoutA mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutA:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_SetSMBusTimeoutA

LL_I2C_GetSMBusTimeoutA

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutA (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTA LL_I2C_GetSMBusTimeoutA

LL_I2C_SetSMBusTimeoutAMode

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutAMode (I2C_TypeDef * I2Cx, uint32_t TimeoutAMode)

Function description

Set the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutAMode:** This parameter can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This bit can only be programmed when TimeoutA is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_SetSMBusTimeoutAMode

LL_I2C_GetSMBusTimeoutAMode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutAMode (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Clock TimeoutA mode.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW
  - LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIDLE LL_I2C_GetSMBusTimeoutAMode

LL_I2C_SetSMBusTimeoutB

Function name

__STATIC_INLINE void LL_I2C_SetSMBusTimeoutB (I2C_TypeDef * I2Cx, uint32_t TimeoutB)

Function description

Configure the SMBus Extended Cumulative Clock TimeoutB (Master or Slave mode).

Parameters

- **I2Cx:** I2C Instance.
- **TimeoutB:** This parameter must be a value between Min_Data=0 and Max_Data=0xFFF.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- These bits can only be programmed when TimeoutB is disabled.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_SetSMBusTimeoutB

LL_I2C_GetSMBusTimeoutB

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusTimeoutB (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Extended Cumulative Clock TimeoutB setting.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMEOUTB LL_I2C_GetSMBusTimeoutB

LL_I2C_EnableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Enable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  – LL_I2C_SMBUS_TIMEOUTA
  – LL_I2C_SMBUS_TIMEOUTB
  – LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_EnableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_EnableSMBusTimeout

LL_I2C_DisableSMBusTimeout

Function name

__STATIC_INLINE void LL_I2C_DisableSMBusTimeout (I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Disable the SMBus Clock Timeout.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  – LL_I2C_SMBUS_TIMEOUTA
  – LL_I2C_SMBUS_TIMEOUTB
  – LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_DisableSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_DisableSMBusTimeout

LL_I2C_IsEnabledSMBusTimeout

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusTimeout (const I2C_TypeDef * I2Cx, uint32_t ClockTimeout)

Function description

Check if the SMBus Clock Timeout is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.
- **ClockTimeout:** This parameter can be one of the following values:
  – LL_I2C_SMBUS_TIMEOUTA
  – LL_I2C_SMBUS_TIMEOUTB
  – LL_I2C_SMBUS_ALL_TIMEOUT

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- TIMEOUTR TIMOUTEN LL_I2C_IsEnabledSMBusTimeout
- TIMEOUTR TEXTEN LL_I2C_IsEnabledSMBusTimeout

LL_I2C_EnableIT_TX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TX (I2C_TypeDef * I2Cx)

Function description

Enable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_EnableIT_TX

LL_I2C_DisableIT_TX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TX (I2C_TypeDef * I2Cx)

Function description

Disable TXIS interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_DisableIT_TX

LL_I2C_IsEnabledIT_TX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TX (const I2C_TypeDef * I2Cx)

Function description

Check if the TXIS Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXIE LL_I2C_IsEnabledIT_TX

LL_I2C_EnableIT_RX

Function name

__STATIC_INLINE void LL_I2C_EnableIT_RX (I2C_TypeDef * I2Cx)

Function description

Enable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_EnableIT_RX

LL_I2C_DisableIT_RX

Function name

__STATIC_INLINE void LL_I2C_DisableIT_RX (I2C_TypeDef * I2Cx)

Function description

Disable RXNE interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_DisableIT_RX

LL_I2C_IsEnabledIT_RX

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_RX (const I2C_TypeDef * I2Cx)

Function description

Check if the RXNE Interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXIE LL_I2C_IsEnabledIT_RX

LL_I2C_EnableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Enable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_EnableIT_ADDR

LL_I2C_DisableIT_ADDR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ADDR (I2C_TypeDef * I2Cx)

Function description

Disable Address match interrupt (slave mode only).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_DisableIT_ADDR

LL_I2C_IsEnabledIT_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ADDR (const I2C_TypeDef * I2Cx)

Function description

Check if Address match interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ADDRIE LL_I2C_IsEnabledIT_ADDR

LL_I2C_EnableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_EnableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Enable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_EnableIT_NACK

LL_I2C_DisableIT_NACK

Function name

__STATIC_INLINE void LL_I2C_DisableIT_NACK (I2C_TypeDef * I2Cx)

Function description

Disable Not acknowledge received interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_DisableIT_NACK

LL_I2C_IsEnabledIT_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_NACK (const I2C_TypeDef * I2Cx)

Function description

Check if Not acknowledge received interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 NACKIE LL_I2C_IsEnabledIT_NACK

LL_I2C_EnableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_EnableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Enable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_EnableIT_STOP

LL_I2C_DisableIT_STOP

Function name

__STATIC_INLINE void LL_I2C_DisableIT_STOP (I2C_TypeDef * I2Cx)

Function description

Disable STOP detection interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_DisableIT_STOP

LL_I2C_IsEnabledIT_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_STOP (const I2C_TypeDef * I2Cx)

Function description

Check if STOP detection interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 STOPIE LL_I2C_IsEnabledIT_STOP

LL_I2C_EnableIT_TC

Function name

__STATIC_INLINE void LL_I2C_EnableIT_TC (I2C_TypeDef * I2Cx)

Function description

Enable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_EnableIT_TC

LL_I2C_DisableIT_TC

Function name

__STATIC_INLINE void LL_I2C_DisableIT_TC (I2C_TypeDef * I2Cx)

Function description

Disable Transfer Complete interrupt.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Any of these events will generate interrupt : Transfer Complete (TC) Transfer Complete Reload (TCR)

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_DisableIT_TC

LL_I2C_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_TC (const I2C_TypeDef * I2Cx)

Function description

Check if Transfer Complete interrupt is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_I2C_IsEnabledIT_TC

LL_I2C_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_EnableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Enable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_EnableIT_ERR

LL_I2C_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2C_DisableIT_ERR (I2C_TypeDef * I2Cx)

Function description

Disable Error interrupts.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- Any of these errors will generate interrupt : Arbitration Loss (ARLO) Bus Error detection (BERR) Overrun/Underrun (OVR) SMBus Timeout detection (TIMEOUT) SMBus PEC error detection (PECERR) SMBus Alert pin event detection (ALERT)

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_DisableIT_ERR

LL_I2C_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledIT_ERR (const I2C_TypeDef * I2Cx)

Function description

Check if Error interrupts are enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ERRIE LL_I2C_IsEnabledIT_ERR

LL_I2C_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXE (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit data register empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_IsActiveFlag_TXE

LL_I2C_IsActiveFlag_TXIS

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TXIS (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transmit interrupt flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When next data is written in Transmit data register. SET: When Transmit data register is empty.

Reference Manual to LL API cross reference:

- ISR TXIS LL_I2C_IsActiveFlag_TXIS

LL_I2C_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_RXNE (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Receive data register not empty flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: When Receive data register is read. SET: When the received data is copied in Receive data register.

Reference Manual to LL API cross reference:

- ISR RXNE LL_I2C_IsActiveFlag_RXNE

LL_I2C_IsActiveFlag_ADDR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ADDR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Address matched flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When the received slave address matched with one of the enabled slave address.

Reference Manual to LL API cross reference:

- ISR ADDR LL_I2C_IsActiveFlag_ADDR

LL_I2C_IsActiveFlag_NACK

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_NACK (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Not Acknowledge received flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a NACK is received after a byte transmission.

Reference Manual to LL API cross reference:

- ISR NACKF LL_I2C_IsActiveFlag_NACK

LL_I2C_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_STOP (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR STOPF LL_I2C_IsActiveFlag_STOP

LL_I2C_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TC (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=0, AUTOEND=0 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TC LL_I2C_IsActiveFlag_TC

LL_I2C_IsActiveFlag_TCR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_TCR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Transfer complete flag (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When RELOAD=1 and NBYTES date have been transferred.

Reference Manual to LL API cross reference:

- ISR TCR LL_I2C_IsActiveFlag_TCR

LL_I2C_IsActiveFlag_BERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BERR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a misplaced Start or Stop condition is detected.

Reference Manual to LL API cross reference:

- ISR BERR LL_I2C_IsActiveFlag_BERR

LL_I2C_IsActiveFlag_ARLO

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_ARLO (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When arbitration lost.

Reference Manual to LL API cross reference:

- ISR ARLO LL_I2C_IsActiveFlag_ARLO

LL_I2C_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_OVR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Overrun/Underrun flag (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When an overrun/underrun error occurs (Clock Stretching Disabled).

Reference Manual to LL API cross reference:

- ISR OVR LL_I2C_IsActiveFlag_OVR

LL_I2C_IsActiveSMBusFlag_PECERR

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_PECERR (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus PEC error flag in reception.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When the received PEC does not match with the PEC register content.

Reference Manual to LL API cross reference:

- ISR PECERR LL_I2C_IsActiveSMBusFlag_PECERR

LL_I2C_IsActiveSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_TIMEOUT (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When a timeout or extended clock timeout occurs.

Reference Manual to LL API cross reference:

- ISR TIMEOUT LL_I2C_IsActiveSMBusFlag_TIMEOUT

LL_I2C_IsActiveSMBusFlag_ALERT

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveSMBusFlag_ALERT (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of SMBus alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- RESET: Clear default value. SET: When SMBus host configuration, SMBus alert enabled and a falling edge event occurs on SMBA pin.

Reference Manual to LL API cross reference:

- ISR ALERT LL_I2C_IsActiveSMBusFlag_ALERT

LL_I2C_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_I2C_IsActiveFlag_BUSY (const I2C_TypeDef * I2Cx)

Function description

Indicate the status of Bus Busy flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- RESET: Clear default value. SET: When a Start condition is detected.

Reference Manual to LL API cross reference:

- ISR BUSY LL_I2C_IsActiveFlag_BUSY

LL_I2C_ClearFlag_ADDR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ADDR (I2C_TypeDef * I2Cx)

Function description

Clear Address Matched flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ADDRCF LL_I2C_ClearFlag_ADDR

LL_I2C_ClearFlag_NACK

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_NACK (I2C_TypeDef * I2Cx)

Function description

Clear Not Acknowledge flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NACKCF LL_I2C_ClearFlag_NACK

LL_I2C_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_STOP (I2C_TypeDef * I2Cx)

Function description

Clear Stop detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR STOPCF LL_I2C_ClearFlag_STOP

LL_I2C_ClearFlag_TXE

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_TXE (I2C_TypeDef * I2Cx)

Function description

Clear Transmit data register empty flag (TXE).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- This bit can be clear by software in order to flush the transmit data register (TXDR).

Reference Manual to LL API cross reference:

- ISR TXE LL_I2C_ClearFlag_TXE

LL_I2C_ClearFlag_BERR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_BERR (I2C_TypeDef * I2Cx)

Function description

Clear Bus error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR BERRCF LL_I2C_ClearFlag_BERR

LL_I2C_ClearFlag_ARLO

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_ARLO (I2C_TypeDef * I2Cx)

Function description

Clear Arbitration lost flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARLOCF LL_I2C_ClearFlag_ARLO

LL_I2C_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2C_ClearFlag_OVR (I2C_TypeDef * I2Cx)

Function description

Clear Overrun/Underrun flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR OVRCF LL_I2C_ClearFlag_OVR

LL_I2C_ClearSMBusFlag_PECERR

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_PECERR (I2C_TypeDef * I2Cx)

Function description

Clear SMBus PEC error flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR PECCF LL_I2C_ClearSMBusFlag_PECERR

LL_I2C_ClearSMBusFlag_TIMEOUT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_TIMEOUT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Timeout detection flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR TIMOUTCF LL_I2C_ClearSMBusFlag_TIMEOUT

LL_I2C_ClearSMBusFlag_ALERT

Function name

__STATIC_INLINE void LL_I2C_ClearSMBusFlag_ALERT (I2C_TypeDef * I2Cx)

Function description

Clear SMBus Alert flag.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- ICR ALERTCF LL_I2C_ClearSMBusFlag_ALERT

LL_I2C_EnableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_EnableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Enable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Automatic end mode : a STOP condition is automatically sent when NBYTES data are transferred. This bit has no effect in slave mode or when RELOAD bit is set.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_EnableAutoEndMode

LL_I2C_DisableAutoEndMode

Function name

__STATIC_INLINE void LL_I2C_DisableAutoEndMode (I2C_TypeDef * I2Cx)

Function description

Disable automatic STOP condition generation (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- Software end mode : TC flag is set when NBYTES data are transferre, stretching SCL low.

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_DisableAutoEndMode

LL_I2C_IsEnabledAutoEndMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAutoEndMode (const I2C_TypeDef * I2Cx)

Function description

Check if automatic STOP condition is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 AUTOEND LL_I2C_IsEnabledAutoEndMode

LL_I2C_EnableReloadMode

Function name

__STATIC_INLINE void LL_I2C_EnableReloadMode (I2C_TypeDef * I2Cx)

Function description

Enable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is not completed after the NBYTES data transfer, NBYTES will be reloaded when TCR flag is set.

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_EnableReloadMode

LL_I2C_DisableReloadMode

Function name

__STATIC_INLINE void LL_I2C_DisableReloadMode (I2C_TypeDef * I2Cx)

Function description

Disable reload mode (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The transfer is completed after the NBYTES data transfer(STOP or RESTART will follow).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_DisableReloadMode

LL_I2C_IsEnabledReloadMode

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledReloadMode (const I2C_TypeDef * I2Cx)

Function description

Check if reload mode is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RELOAD LL_I2C_IsEnabledReloadMode

LL_I2C_SetTransferSize

Function name

__STATIC_INLINE void LL_I2C_SetTransferSize (I2C_TypeDef * I2Cx, uint32_t TransferSize)

Function description

Configure the number of bytes for transfer.

Parameters

- **I2Cx:** I2C Instance.
- **TransferSize:** This parameter must be a value between Min_Data=0x00 and Max_Data=0xFF.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_SetTransferSize

LL_I2C_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferSize (const I2C_TypeDef * I2Cx)

Function description

Get the number of bytes configured for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- CR2 NBYTES LL_I2C_GetTransferSize

LL_I2C_AcknowledgeNextData

Function name

__STATIC_INLINE void LL_I2C_AcknowledgeNextData (I2C_TypeDef * I2Cx, uint32_t TypeAcknowledge)

Function description

Prepare the generation of a ACKnowledge or Non ACKnowledge condition after the address receive match code or next received byte.

Parameters

- **I2Cx:** I2C Instance.
- **TypeAcknowledge:** This parameter can be one of the following values:
  – LL_I2C_ACK
  – LL_I2C_NACK

Return values

- **None:**

Notes

- Usage in Slave mode only.

Reference Manual to LL API cross reference:

- CR2 NACK LL_I2C_AcknowledgeNextData

LL_I2C_GenerateStartCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStartCondition (I2C_TypeDef * I2Cx)

Function description

Generate a START or RESTART condition.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The START bit can be set even if bus is BUSY or I2C is in slave mode. This action has no effect when RELOAD is set.

Reference Manual to LL API cross reference:

- CR2 START LL_I2C_GenerateStartCondition

LL_I2C_GenerateStopCondition

Function name

__STATIC_INLINE void LL_I2C_GenerateStopCondition (I2C_TypeDef * I2Cx)

Function description

Generate a STOP condition after the current byte transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_I2C_GenerateStopCondition

LL_I2C_EnableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_EnableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Enable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master sends the complete 10bit slave address read sequence : Start + 2 bytes 10bit address in Write direction + Restart + first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_EnableAuto10BitRead

LL_I2C_DisableAuto10BitRead

Function name

__STATIC_INLINE void LL_I2C_DisableAuto10BitRead (I2C_TypeDef * I2Cx)

Function description

Disable automatic RESTART Read request condition for 10bit address header (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The master only sends the first 7 bits of 10bit address in Read direction.

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_DisableAuto10BitRead

LL_I2C_IsEnabledAuto10BitRead

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledAuto10BitRead (const I2C_TypeDef * I2Cx)

Function description

Check if automatic RESTART Read request condition for 10bit address header is enabled or disabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 HEAD10R LL_I2C_IsEnabledAuto10BitRead

LL_I2C_SetTransferRequest

Function name

__STATIC_INLINE void LL_I2C_SetTransferRequest (I2C_TypeDef * I2Cx, uint32_t TransferRequest)

Function description

Configure the transfer direction (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **TransferRequest:** This parameter can be one of the following values:
  – LL_I2C_REQUEST_WRITE
  – LL_I2C_REQUEST_READ

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_SetTransferRequest

LL_I2C_GetTransferRequest

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferRequest (const I2C_TypeDef * I2Cx)

Function description

Get the transfer direction requested (master mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  – LL_I2C_REQUEST_WRITE
  – LL_I2C_REQUEST_READ

Reference Manual to LL API cross reference:

- CR2 RD_WRN LL_I2C_GetTransferRequest

LL_I2C_SetSlaveAddr

Function name

__STATIC_INLINE void LL_I2C_SetSlaveAddr (I2C_TypeDef * I2Cx, uint32_t SlaveAddr)

Function description

Configure the slave address for transfer (master mode).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** This parameter must be a value between Min_Data=0x00 and Max_Data=0x3F.

Return values

- **None:**

Notes

- Changing these bits when START bit is set is not allowed.

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_SetSlaveAddr

LL_I2C_GetSlaveAddr

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSlaveAddr (const I2C_TypeDef * I2Cx)

Function description

Get the slave address programmed for transfer.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_GetSlaveAddr

LL_I2C_HandleTransfer

Function name

__STATIC_INLINE void LL_I2C_HandleTransfer (I2C_TypeDef * I2Cx, uint32_t SlaveAddr, uint32_t SlaveAddrSize, uint32_t TransferSize, uint32_t EndMode, uint32_t Request)

Function description

Handles I2Cx communication when starting transfer or during transfer (TC or TCR flag are set).

Parameters

- **I2Cx:** I2C Instance.
- **SlaveAddr:** Specifies the slave address to be programmed.
- **SlaveAddrSize:** This parameter can be one of the following values:
  – LL_I2C_ADDRSLAVE_7BIT
  – LL_I2C_ADDRSLAVE_10BIT
- **TransferSize:** Specifies the number of bytes to be programmed. This parameter must be a value between Min_Data=0 and Max_Data=255.
- **EndMode:** This parameter can be one of the following values:
  – LL_I2C_MODE_RELOAD
  – LL_I2C_MODE_AUTOEND
  – LL_I2C_MODE_SOFTEND
  – LL_I2C_MODE_SMBUS_RELOAD
  – LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC
  – LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC
  – LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC
  – LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC
- **Request:** This parameter can be one of the following values:
  – LL_I2C_GENERATE_NOSTARTSTOP
  – LL_I2C_GENERATE_STOP
  – LL_I2C_GENERATE_START_READ
  – LL_I2C_GENERATE_START_WRITE
  – LL_I2C_GENERATE_RESTART_7BIT_READ
  – LL_I2C_GENERATE_RESTART_7BIT_WRITE
  – LL_I2C_GENERATE_RESTART_10BIT_READ
  – LL_I2C_GENERATE_RESTART_10BIT_WRITE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SADD LL_I2C_HandleTransfer
- CR2 ADD10 LL_I2C_HandleTransfer
- CR2 RD_WRN LL_I2C_HandleTransfer
- CR2 START LL_I2C_HandleTransfer
- CR2 STOP LL_I2C_HandleTransfer
- CR2 RELOAD LL_I2C_HandleTransfer
- CR2 NBYTES LL_I2C_HandleTransfer
- CR2 AUTOEND LL_I2C_HandleTransfer
- CR2 HEAD10R LL_I2C_HandleTransfer

LL_I2C_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTransferDirection (const I2C_TypeDef * I2Cx)

Function description

Indicate the value of transfer direction (slave mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  – LL_I2C_DIRECTION_WRITE
  – LL_I2C_DIRECTION_READ

Notes

- RESET: Write transfer, Slave enters in receiver mode. SET: Read transfer, Slave enters in transmitter mode.

Reference Manual to LL API cross reference:

- ISR DIR LL_I2C_GetTransferDirection

LL_I2C_GetAddressMatchCode

Function name

__STATIC_INLINE uint32_t LL_I2C_GetAddressMatchCode (const I2C_TypeDef * I2Cx)

Function description

Return the slave matched address.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- ISR ADDCODE LL_I2C_GetAddressMatchCode

LL_I2C_EnableSMBusPECCompare

Function name

__STATIC_INLINE void LL_I2C_EnableSMBusPECCompare (I2C_TypeDef * I2Cx)

Function description

Enable internal comparison of the SMBus Packet Error byte (transmission or reception mode).

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.
- This feature is cleared by hardware when the PEC byte is transferred, or when a STOP condition or an Address Matched is received. This bit has no effect when RELOAD bit is set. This bit has no effect in device mode when SBC bit is not set.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_EnableSMBusPECCompare

LL_I2C_IsEnabledSMBusPECCompare

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabledSMBusPECCompare (const I2C_TypeDef * I2Cx)

Function description

Check if the SMBus Packet Error byte internal comparison is requested or not.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- CR2 PECBYTE LL_I2C_IsEnabledSMBusPECCompare

LL_I2C_GetSMBusPEC

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSMBusPEC (const I2C_TypeDef * I2Cx)

Function description

Get the SMBus Packet Error byte calculated.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Notes

- The macro IS_SMBUS_ALL_INSTANCE(I2Cx) can be used to check whether or not SMBus feature is supported by the I2Cx Instance.

Reference Manual to LL API cross reference:

- PECR PEC LL_I2C_GetSMBusPEC

LL_I2C_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_I2C_ReceiveData8 (const I2C_TypeDef * I2Cx)

Function description

Read Receive Data register.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RXDR RXDATA LL_I2C_ReceiveData8

LL_I2C_TransmitData8

Function name

__STATIC_INLINE void LL_I2C_TransmitData8 (I2C_TypeDef * I2Cx, uint8_t Data)

Function description

Write in Transmit Data Register .

Parameters

- **I2Cx:** I2C Instance.
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR TXDATA LL_I2C_TransmitData8

LL_I2C_Enable_SelectedTrigger

Function name

__STATIC_INLINE void LL_I2C_Enable_SelectedTrigger (I2C_TypeDef * I2Cx)

Function description

Enable Selected Trigger.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_I2C_Enable_SelectedTrigger

LL_I2C_Disable_SelectedTrigger

Function name

__STATIC_INLINE void LL_I2C_Disable_SelectedTrigger (I2C_TypeDef * I2Cx)

Function description

Disable Selected Trigger.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_I2C_Disable_SelectedTrigger

LL_I2C_IsEnabled_SelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_I2C_IsEnabled_SelectedTrigger (const I2C_TypeDef * I2Cx)

Function description

Indicate if selected Trigger is disabled or enabled.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_I2C_IsEnabled_SelectedTrigger

LL_I2C_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_I2C_SetTriggerPolarity (I2C_TypeDef * I2Cx, uint32_t Polarity)

Function description

Set the trigger polarity.

Parameters

- **I2Cx:** I2C Instance.
- **Polarity:** This parameter can be one of the following values:
  - LL_I2C_TRIG_POLARITY_RISING
  - LL_I2C_TRIG_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_I2C_SetTriggerPolarity

LL_I2C_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_I2C_GetTriggerPolarity (const I2C_TypeDef * I2Cx)

Function description

Get the trigger polarity.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_TRIG_POLARITY_RISING
  - LL_I2C_TRIG_POLARITY_FALLING

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_I2C_GetTriggerPolarity

LL_I2C_SetSelectedTrigger

Function name

__STATIC_INLINE void LL_I2C_SetSelectedTrigger (I2C_TypeDef * I2Cx, uint32_t Trigger)

Function description

Set the selected trigger.

Parameters

- **I2Cx:** I2C Instance.
- **Trigger:** This parameter can be one of the following values:
  - LL_I2C_GRP1_GPDMA_CH0_TCF_TRG
  - LL_I2C_GRP1_GPDMA_CH1_TCF_TRG
  - LL_I2C_GRP1_GPDMA_CH2_TCF_TRG
  - LL_I2C_GRP1_GPDMA_CH3_TCF_TRG
  - LL_I2C_GRP1_EXTI5_TRG
  - LL_I2C_GRP1_EXTI9_TRG
  - LL_I2C_GRP1_LPTIM1_CH1_TRG
  - LL_I2C_GRP1_LPTIM2_CH1_TRG
  - LL_I2C_GRP1_COMP1_TRG
  - LL_I2C_GRP1_COMP2_TRG
  - LL_I2C_GRP1_RTC_ALRA_TRG
  - LL_I2C_GRP1_RTC_WUT_TRG
  - LL_I2C_GRP2_LPDMA_CH0_TCF_TRG
  - LL_I2C_GRP2_LPDMA_CH1_TCF_TRG
  - LL_I2C_GRP2_LPDMA_CH2_TCF_TRG
  - LL_I2C_GRP2_LPDMA_CH3_TCF_TRG
  - LL_I2C_GRP2_EXTI5_TRG
  - LL_I2C_GRP2_EXTI8_TRG
  - LL_I2C_GRP2_LPTIM1_CH1_TRG
  - LL_I2C_GRP2_LPTIM3_CH1_TRG
  - LL_I2C_GRP2_COMP1_TRG
  - LL_I2C_GRP2_COMP2_TRG
  - LL_I2C_GRP2_RTC_ALRA_TRG
  - LL_I2C_GRP2_RTC_WUT_TRG

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_I2C_SetSelectedTrigger

LL_I2C_GetSelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_I2C_GetSelectedTrigger (const I2C_TypeDef * I2Cx)

Function description

Get the selected trigger.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_I2C_GRP1_GPDMA_CH0_TCF_TRG
  - LL_I2C_GRP1_GPDMA_CH1_TCF_TRG
  - LL_I2C_GRP1_GPDMA_CH2_TCF_TRG
  - LL_I2C_GRP1_GPDMA_CH3_TCF_TRG
  - LL_I2C_GRP1_EXTI5_TRG
  - LL_I2C_GRP1_EXTI9_TRG
  - LL_I2C_GRP1_LPTIM1_CH1_TRG
  - LL_I2C_GRP1_LPTIM2_CH1_TRG
  - LL_I2C_GRP1_COMP1_TRG
  - LL_I2C_GRP1_COMP2_TRG
  - LL_I2C_GRP1_RTC_ALRA_TRG
  - LL_I2C_GRP1_RTC_WUT_TRG
  - LL_I2C_GRP2_LPDMA_CH0_TCF_TRG
  - LL_I2C_GRP2_LPDMA_CH1_TCF_TRG
  - LL_I2C_GRP2_LPDMA_CH2_TCF_TRG
  - LL_I2C_GRP2_LPDMA_CH3_TCF_TRG
  - LL_I2C_GRP2_EXTI5_TRG
  - LL_I2C_GRP2_EXTI8_TRG
  - LL_I2C_GRP2_LPTIM1_CH1_TRG
  - LL_I2C_GRP2_LPTIM3_CH1_TRG
  - LL_I2C_GRP2_COMP1_TRG
  - LL_I2C_GRP2_COMP2_TRG
  - LL_I2C_GRP2_RTC_ALRA_TRG
  - LL_I2C_GRP2_RTC_WUT_TRG

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_I2C_GetSelectedTrigger

LL_I2C_Init

Function name

ErrorStatus LL_I2C_Init (I2C_TypeDef * I2Cx, const LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Initialize the I2C registers according to the specified parameters in I2C_InitStruct.

Parameters

- **I2Cx:** I2C Instance.
- **I2C_InitStruct:** pointer to a LL_I2C_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: I2C registers are initialized
  - ERROR: Not applicable

LL_I2C_DeInit

Function name

ErrorStatus LL_I2C_DeInit (const I2C_TypeDef * I2Cx)

Function description

De-initialize the I2C registers to their default reset values.

Parameters

- **I2Cx:** I2C Instance.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: I2C registers are de-initialized
  – ERROR: I2C registers are not de-initialized

LL_I2C_StructInit

Function name

void LL_I2C_StructInit (LL_I2C_InitTypeDef * I2C_InitStruct)

Function description

Set each LL_I2C_InitTypeDef field to default value.

Parameters

- **I2C_InitStruct:** Pointer to a LL_I2C_InitTypeDef structure.

Return values

- **None:**

105.3 I2C Firmware driver defines

The following section lists the various define and macros of the module.

105.3.1 I2C

I2C

Master Addressing Mode

LL_I2C_ADDRESSING_MODE_7BIT

Master operates in 7-bit addressing mode.

LL_I2C_ADDRESSING_MODE_10BIT

Master operates in 10-bit addressing mode.

Slave Address Length

LL_I2C_ADDRSLAVE_7BIT

Slave Address in 7-bit.

LL_I2C_ADDRSLAVE_10BIT

Slave Address in 10-bit.

Analog Filter Selection

LL_I2C_ANALOGFILTER_ENABLE

Analog filter is enabled.

LL_I2C_ANALOGFILTER_DISABLE

Analog filter is disabled.

Autonomous Trigger Polarity

LL_I2C_TRIG_POLARITY_RISING

I2C triggered on rising edge

LL_I2C_TRIG_POLARITY_FALLING

I2C triggered on falling edge

Autonomous Trigger selection

LL_I2C_TRIG_GRP1

Trigger Group for I2C1, I2C2, I2C4, I2C5, I2C6 (depends on Product)

LL_I2C_TRIG_GRP2

Trigger Group for I2C3

LL_I2C_GRP1_GPDMA_CH0_TCF_TRG

HW Trigger signal is GPDMA_CH0_TRG

LL_I2C_GRP1_GPDMA_CH1_TCF_TRG

HW Trigger signal is GPDMA_CH1_TRG

LL_I2C_GRP1_GPDMA_CH2_TCF_TRG

HW Trigger signal is GPDMA_CH2_TRG

LL_I2C_GRP1_GPDMA_CH3_TCF_TRG

HW Trigger signal is GPDMA_CH3_TRG

LL_I2C_GRP1_EXTI5_TRG

HW Trigger signal is EXTI5_TRG

LL_I2C_GRP1_EXTI9_TRG

HW Trigger signal is EXTI9_TRG

LL_I2C_GRP1_LPTIM1_CH1_TRG

HW Trigger signal is LPTIM1_CH1_TRG

LL_I2C_GRP1_LPTIM2_CH1_TRG

HW Trigger signal is LPTIM2_CH1_TRG

LL_I2C_GRP1_COMP1_TRG

HW Trigger signal is COMP1_TRG

LL_I2C_GRP1_COMP2_TRG

HW Trigger signal is COMP2_TRG

LL_I2C_GRP1_RTC_ALRA_TRG

HW Trigger signal is RTC_ALRA_TRG

LL_I2C_GRP1_RTC_WUT_TRG

HW Trigger signal is RTC_WUT_TRG

LL_I2C_GRP2_LPDMA_CH0_TCF_TRG

HW Trigger signal is LPDMA_CH0_TRG

LL_I2C_GRP2_LPDMA_CH1_TCF_TRG

HW Trigger signal is LPDMA_CH1_TRG

LL_I2C_GRP2_LPDMA_CH2_TCF_TRG

HW Trigger signal is LPDMA_CH2_TRG

LL_I2C_GRP2_LPDMA_CH3_TCF_TRG

HW Trigger signal is LPDMA_CH3_TRG

LL_I2C_GRP2_EXTI5_TRG

HW Trigger signal is EXTI5_TRG

LL_I2C_GRP2_EXTI8_TRG

HW Trigger signal is EXTI8_TRG

LL_I2C_GRP2_LPTIM1_CH1_TRG

HW Trigger signal is LPTIM1_CH1_TRG

LL_I2C_GRP2_LPTIM3_CH1_TRG

HW Trigger signal is LPTIM3_CH1_TRG

LL_I2C_GRP2_COMP1_TRG

HW Trigger signal is COMP1_TRG

LL_I2C_GRP2_COMP2_TRG

HW Trigger signal is COMP2_TRG

LL_I2C_GRP2_RTC_ALRA_TRG

HW Trigger signal is RTC_ALRA_TRG

LL_I2C_GRP2_RTC_WUT_TRG

HW Trigger signal is RTC_WUT_TRG

Clear Flags Defines

LL_I2C_ICR_ADDRCF

Address Matched flag

LL_I2C_ICR_NACKCF

Not Acknowledge flag

LL_I2C_ICR_STOPCF

Stop detection flag

LL_I2C_ICR_BERRCF

Bus error flag

LL_I2C_ICR_ARLOCF

Arbitration Lost flag

LL_I2C_ICR_OVRCF

Overrun/Underrun flag

LL_I2C_ICR_PECCF

PEC error flag

LL_I2C_ICR_TIMOUTCF

Timeout detection flag

LL_I2C_ICR_ALERTCF

Alert flag

Read Write Direction

LL_I2C_DIRECTION_WRITE

Write transfer request by master, slave enters receiver mode.

LL_I2C_DIRECTION_READ

Read transfer request by master, slave enters transmitter mode.

DMA Register Data

LL_I2C_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_I2C_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

Start And Stop Generation

LL_I2C_GENERATE_NOSTARTSTOP

Don't Generate Stop and Start condition.

LL_I2C_GENERATE_STOP

Generate Stop condition (Size should be set to 0).

LL_I2C_GENERATE_START_READ

Generate Start for read request.

LL_I2C_GENERATE_START_WRITE

Generate Start for write request.

LL_I2C_GENERATE_RESTART_7BIT_READ

Generate Restart for read request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_7BIT_WRITE

Generate Restart for write request, slave 7Bit address.

LL_I2C_GENERATE_RESTART_10BIT_READ

Generate Restart for read request, slave 10Bit address.

LL_I2C_GENERATE_RESTART_10BIT_WRITE

Generate Restart for write request, slave 10Bit address.

Get Flags Defines

LL_I2C_ISR_TXE

Transmit data register empty

LL_I2C_ISR_TXIS

Transmit interrupt status

LL_I2C_ISR_RXNE

Receive data register not empty

LL_I2C_ISR_ADDR

Address matched (slave mode)

LL_I2C_ISR_NACKF

Not Acknowledge received flag

LL_I2C_ISR_STOPF

Stop detection flag

LL_I2C_ISR_TC

Transfer Complete (master mode)

LL_I2C_ISR_TCR

Transfer Complete Reload

LL_I2C_ISR_BERR

Bus error

LL_I2C_ISR_ARLO

Arbitration lost

LL_I2C_ISR_OVR

Overrun/Underrun (slave mode)

LL_I2C_ISR_PECERR

PEC Error in reception (SMBus mode)

LL_I2C_ISR_TIMEOUT

Timeout detection flag (SMBus mode)

LL_I2C_ISR_ALERT

SMBus alert (SMBus mode)

LL_I2C_ISR_BUSY

Bus busy

Acknowledge Generation

LL_I2C_ACK

ACK is sent after current received byte.

LL_I2C_NACK

NACK is sent after current received byte.

IT Defines

LL_I2C_CR1_TXIE

TX Interrupt enable

LL_I2C_CR1_RXIE

RX Interrupt enable

LL_I2C_CR1_ADDRIE

Address match Interrupt enable (slave only)

LL_I2C_CR1_NACKIE

Not acknowledge received Interrupt enable

LL_I2C_CR1_STOPIE

STOP detection Interrupt enable

LL_I2C_CR1_TCIE

Transfer Complete interrupt enable

LL_I2C_CR1_ERRIE

Error interrupts enable

Transfer End Mode

LL_I2C_MODE_RELOAD

Enable I2C Reload mode.

LL_I2C_MODE_AUTOEND

Enable I2C Automatic end mode with no HW PEC comparison.

LL_I2C_MODE_SOFTEND

Enable I2C Software end mode with no HW PEC comparison.

LL_I2C_MODE_SMBUS_RELOAD

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_NO_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_NO_PEC

Enable SMBUS Software end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_AUTOEND_WITH_PEC

Enable SMBUS Automatic end mode with HW PEC comparison.

LL_I2C_MODE_SMBUS_SOFTEND_WITH_PEC

Enable SMBUS Software end mode with HW PEC comparison.

Own Address 1 Length

LL_I2C_OWNADDRESS1_7BIT

Own address 1 is a 7-bit address.

LL_I2C_OWNADDRESS1_10BIT

Own address 1 is a 10-bit address.

Own Address 2 Masks

LL_I2C_OWNADDRESS2_NOMASK

Own Address2 No mask.

LL_I2C_OWNADDRESS2_MASK01

Only Address2 bits[7:2] are compared.

LL_I2C_OWNADDRESS2_MASK02

Only Address2 bits[7:3] are compared.

LL_I2C_OWNADDRESS2_MASK03

Only Address2 bits[7:4] are compared.

LL_I2C_OWNADDRESS2_MASK04

Only Address2 bits[7:5] are compared.

LL_I2C_OWNADDRESS2_MASK05

Only Address2 bits[7:6] are compared.

LL_I2C_OWNADDRESS2_MASK06

Only Address2 bits[7] are compared.

LL_I2C_OWNADDRESS2_MASK07

No comparison is done. All Address2 are acknowledged.

Peripheral Mode

LL_I2C_MODE_I2C

I2C Master or Slave mode

LL_I2C_MODE_SMBUS_HOST

SMBus Host address acknowledge

LL_I2C_MODE_SMBUS_DEVICE

SMBus Device default mode (Default address not acknowledge)

LL_I2C_MODE_SMBUS_DEVICE_ARP

SMBus Device Default address acknowledge

Transfer Request Direction

LL_I2C_REQUEST_WRITE

Master request a write transfer.

LL_I2C_REQUEST_READ

Master request a read transfer.

SMBus TimeoutA Mode SCL SDA Timeout

LL_I2C_SMBUS_TIMEOUTA_MODE_SCL_LOW

TimeoutA is used to detect SCL low level timeout.

LL_I2C_SMBUS_TIMEOUTA_MODE_SDA_SCL_HIGH

TimeoutA is used to detect both SCL and SDA high level timeout.

SMBus Timeout Selection

LL_I2C_SMBUS_TIMEOUTA

TimeoutA enable bit

LL_I2C_SMBUS_TIMEOUTB

TimeoutB (extended clock) enable bit

LL_I2C_SMBUS_ALL_TIMEOUT

TimeoutA and TimeoutB (extended clock) enable bits

Convert SDA SCL timings

__LL_I2C_CONVERT_TIMINGS

Description:

- Configure the SDA setup, hold time and the SCL high, low period.

Parameters:

- __PRESCALER__: This parameter must be a value between Min_Data=0 and Max_Data=0xF.
- __SETUP_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tscldel = (SCLDEL+1)xtpresc)
- __HOLD_TIME__: This parameter must be a value between Min_Data=0 and Max_Data=0xF. (tsdadel = SDADELxtpresc)
- __SCLH_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tsclh = (SCLH+1)xtpresc)
- __SCLL_PERIOD__: This parameter must be a value between Min_Data=0 and Max_Data=0xFF. (tscll = (SCLL+1)xtpresc)

Return value:

- Value: between Min_Data=0 and Max_Data=0xFFFFFFFF

Common Write and read registers Macros

LL_I2C_WriteReg

Description:

- Write a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2C_ReadReg

Description:

- Read a value in I2C register.

Parameters:

- __INSTANCE__: I2C Instance
- __REG__: Register to be read

Return value:

- Register: value

106 LL ICACHE Generic Driver

106.1 ICACHE Firmware driver registers structures

106.1.1 LL_ICACHE_RegionTypeDef

LL_ICACHE_RegionTypeDef is defined in the stm32u5xx_ll_icache.h

Data Fields

- ***uint32_t BaseAddress***
- ***uint32_t RemapAddress***
- ***uint32_t Size***
- ***uint32_t TrafficRoute***
- ***uint32_t OutputBurstType***

Field Documentation

- ***uint32_t LL_ICACHE_RegionTypeDef::BaseAddress***
  Configures the C-AHB base address to be remapped
- ***uint32_t LL_ICACHE_RegionTypeDef::RemapAddress***
  Configures the remap address to be remapped
- ***uint32_t LL_ICACHE_RegionTypeDef::Size***
  Configures the region size. This parameter can be a value of ***ICACHE_LL_EC_Region_Size***
- ***uint32_t LL_ICACHE_RegionTypeDef::TrafficRoute***
  Selects the traffic route. This parameter can be a value of ***ICACHE_LL_EC_Traffic_Route***
- ***uint32_t LL_ICACHE_RegionTypeDef::OutputBurstType***
  Selects the output burst type. This parameter can be a value of ***ICACHE_LL_EC_Output_Burst_Type***

106.2 ICACHE Firmware driver API description

The following section lists the various functions of the ICACHE library.

106.2.1 Detailed description of functions

LL_ICACHE_Enable

Function name

__STATIC_INLINE void LL_ICACHE_Enable (void)

Function description

Enable the ICACHE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_Enable

LL_ICACHE_Disable

Function name

__STATIC_INLINE void LL_ICACHE_Disable (void)

Function description

Disable the ICACHE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_Disable

LL_ICACHE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabled (void)

Function description

Return if ICACHE is enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_ICACHE_IsEnabled

LL_ICACHE_SetMode

Function name

__STATIC_INLINE void LL_ICACHE_SetMode (uint32_t Mode)

Function description

Select the ICACHE operating mode.

Parameters

- **Mode:** This parameter can be one of the following values:
  - LL_ICACHE_1WAY
  - LL_ICACHE_2WAYS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR WAYSEL LL_ICACHE_SetMode

LL_ICACHE_GetMode

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetMode (void)

Function description

Get the selected ICACHE operating mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_1WAY
  - LL_ICACHE_2WAYS

Reference Manual to LL API cross reference:

- CR WAYSEL LL_ICACHE_GetMode

LL_ICACHE_Invalidate

Function name

__STATIC_INLINE void LL_ICACHE_Invalidate (void)

Function description

Invalidate the ICACHE.

Return values

- **None:**

Notes

- Until the BSYEND flag is set, the cache is bypassed.

Reference Manual to LL API cross reference:

- CR CACHEINV LL_ICACHE_Invalidate

LL_ICACHE_EnableMonitors

Function name

__STATIC_INLINE void LL_ICACHE_EnableMonitors (uint32_t Monitors)

Function description

Enable the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_EnableMonitors
- CR MISSMEN LL_ICACHE_EnableMonitors

LL_ICACHE_DisableMonitors

Function name

__STATIC_INLINE void LL_ICACHE_DisableMonitors (uint32_t Monitors)

Function description

Disable the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_DisableMonitors
- CR MISSMEN LL_ICACHE_DisableMonitors

LL_ICACHE_IsEnabledMonitors

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledMonitors (uint32_t Monitors)

Function description

Check if the monitor(s) is(are) enabled or disabled.

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **State:** of parameter value (1 or 0).

Reference Manual to LL API cross reference:

- CR HITMEN LL_ICACHE_IsEnabledMonitors
- CR MISSMEN LL_ICACHE_IsEnabledMonitors

LL_ICACHE_ResetMonitors

Function name

__STATIC_INLINE void LL_ICACHE_ResetMonitors (uint32_t Monitors)

Function description

Reset the hit/miss monitor(s).

Parameters

- **Monitors:** This parameter can be one or a combination of the following values:
  - LL_ICACHE_MONITOR_HIT
  - LL_ICACHE_MONITOR_MISS
  - LL_ICACHE_MONITOR_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HITMRST LL_ICACHE_ResetMonitors
- CR MISSMRST LL_ICACHE_ResetMonitors

LL_ICACHE_GetHitMonitor

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetHitMonitor (void)

Function description

Get the Hit monitor.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFFFFFF

Notes

- Upon reaching the 32-bit maximum value, hit monitor does not wrap.

Reference Manual to LL API cross reference:

- HMONR HITMON LL_ICACHE_GetHitMonitor

LL_ICACHE_GetMissMonitor

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetMissMonitor (void)

Function description

Get the Miss monitor.

Return values

- **Value:** between Min_Data=0 and Max_Data=0xFFFF

Notes

- Upon reaching the 16-bit maximum value, miss monitor does not wrap.

Reference Manual to LL API cross reference:

- MMONR MISSMON LL_ICACHE_GetMissMonitor

LL_ICACHE_EnableIT_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_EnableIT_BSYEND (void)

Function description

Enable BSYEND interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_EnableIT_BSYEND

LL_ICACHE_DisableIT_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_DisableIT_BSYEND (void)

Function description

Disable BSYEND interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_DisableIT_BSYEND

LL_ICACHE_IsEnabledIT_BSYEND

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledIT_BSYEND (void)

Function description

Check if the BSYEND Interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER BSYENDIE LL_ICACHE_IsEnabledIT_BSYEND

LL_ICACHE_EnableIT_ERR

Function name

__STATIC_INLINE void LL_ICACHE_EnableIT_ERR (void)

Function description

Enable ERR interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_EnableIT_ERR

LL_ICACHE_DisableIT_ERR

Function name

__STATIC_INLINE void LL_ICACHE_DisableIT_ERR (void)

Function description

Disable ERR interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_DisableIT_ERR

LL_ICACHE_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledIT_ERR (void)

Function description

Check if the ERR Interrupt is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ERRIE LL_ICACHE_IsEnabledIT_ERR

LL_ICACHE_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_BUSY (void)

Function description

Indicate the status of an ongoing operation flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSYF LL_ICACHE_IsActiveFlag_BUSY

LL_ICACHE_IsActiveFlag_BSYEND

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_BSYEND (void)

Function description

Indicate the status of an operation end flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSYEND LL_ICACHE_IsActiveFlag_BSYEND

LL_ICACHE_IsActiveFlag_ERR

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsActiveFlag_ERR (void)

Function description

Indicate the status of an error flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ERRF LL_ICACHE_IsActiveFlag_ERR

LL_ICACHE_ClearFlag_BSYEND

Function name

__STATIC_INLINE void LL_ICACHE_ClearFlag_BSYEND (void)

Function description

Clear busy end of operation flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR CBSYENDF LL_ICACHE_ClearFlag_BSYEND

LL_ICACHE_ClearFlag_ERR

Function name

__STATIC_INLINE void LL_ICACHE_ClearFlag_ERR (void)

Function description

Clear error flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FCR ERRF LL_ICACHE_ClearFlag_ERR

LL_ICACHE_EnableRegion

Function name

__STATIC_INLINE void LL_ICACHE_EnableRegion (uint32_t Region)

Function description

Enable the remapped memory region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **None:**

Notes

- The region must have been already configured.

Reference Manual to LL API cross reference:

- CRRx REN LL_ICACHE_EnableRegion

LL_ICACHE_DisableRegion

Function name

__STATIC_INLINE void LL_ICACHE_DisableRegion (uint32_t Region)

Function description

Disable the remapped memory region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx REN LL_ICACHE_DisableRegion

LL_ICACHE_IsEnabledRegion

Function name

__STATIC_INLINE uint32_t LL_ICACHE_IsEnabledRegion (uint32_t Region)

Function description

Return if remapped memory region is enabled or not.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CRRx REN LL_ICACHE_IsEnabledRegion

LL_ICACHE_SetRegionBaseAddress

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionBaseAddress (uint32_t Region, uint32_t Address)

Function description

Select the memory remapped region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Address:** Alias address in the Code region

Return values

- **None:**

Notes

- The useful bits depends on RSIZE as described in the Reference Manual.

Reference Manual to LL API cross reference:

- CRRx BASEADDR LL_ICACHE_SetRegionBaseAddress

LL_ICACHE_GetRegionBaseAddress

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionBaseAddress (uint32_t Region)

Function description

Get the memory remapped region base address.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Address:** Alias address in the Code region

Notes

- The base address is the alias in the Code region.
- The useful bits depends on RSIZE as described in the Reference Manual.

Reference Manual to LL API cross reference:

- CRRx BASEADDR LL_ICACHE_GetRegionBaseAddress

LL_ICACHE_SetRegionRemapAddress

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionRemapAddress (uint32_t Region, uint32_t Address)

Function description

Select the memory remapped region address.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_ICACHE_REGION_0
  – LL_ICACHE_REGION_1
  – LL_ICACHE_REGION_2
  – LL_ICACHE_REGION_3
- **Address:** Memory address to remap

Return values

- **None:**

Notes

- The useful bits depends on RSIZE as described in the Reference Manual.

Reference Manual to LL API cross reference:

- CRRx REMAPADDR LL_ICACHE_SetRegionRemapAddress

LL_ICACHE_GetRegionRemapAddress

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionRemapAddress (uint32_t Region)

Function description

Get the memory remapped region address.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_ICACHE_REGION_0
  – LL_ICACHE_REGION_1
  – LL_ICACHE_REGION_2
  – LL_ICACHE_REGION_3

Return values

- **Address:** Remapped memory address

Notes

- The useful bits depends on RSIZE as described in the Reference Manual.

Reference Manual to LL API cross reference:

- CRRx REMAPADDR LL_ICACHE_GetRegionRemapAddress

LL_ICACHE_SetRegionSize

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionSize (uint32_t Region, uint32_t Size)

Function description

Select the memory remapped region size.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **Size:** This parameter can be one of the following values:
  - LL_ICACHE_REGIONSIZE_2MB
  - LL_ICACHE_REGIONSIZE_4MB
  - LL_ICACHE_REGIONSIZE_8MB
  - LL_ICACHE_REGIONSIZE_16MB
  - LL_ICACHE_REGIONSIZE_32MB
  - LL_ICACHE_REGIONSIZE_64MB
  - LL_ICACHE_REGIONSIZE_128MB

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx RSIZE LL_ICACHE_SetRegionSize

LL_ICACHE_GetRegionSize

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionSize (uint32_t Region)

Function description

Get the selected the memory remapped region size.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3

Return values

- **Returned:** value can be one of the following values:
  - LL_ICACHE_REGIONSIZE_2MB
  - LL_ICACHE_REGIONSIZE_4MB
  - LL_ICACHE_REGIONSIZE_8MB
  - LL_ICACHE_REGIONSIZE_16MB
  - LL_ICACHE_REGIONSIZE_32MB
  - LL_ICACHE_REGIONSIZE_64MB
  - LL_ICACHE_REGIONSIZE_128MB

Reference Manual to LL API cross reference:

- CRRx RSIZE LL_ICACHE_GetRegionSize

LL_ICACHE_SetRegionOutputBurstType

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionOutputBurstType (uint32_t Region, uint32_t Type)

Function description

Select the memory remapped region output burst type.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_ICACHE_REGION_0
  – LL_ICACHE_REGION_1
  – LL_ICACHE_REGION_2
  – LL_ICACHE_REGION_3
- **Type:** This parameter can be one of the following values:
  – LL_ICACHE_OUTPUT_BURST_WRAP
  – LL_ICACHE_OUTPUT_BURST_INCR

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx HBURST LL_ICACHE_SetRegionOutputBurstType

LL_ICACHE_GetRegionOutputBurstType

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionOutputBurstType (uint32_t Region)

Function description

Get the selected the memory remapped region output burst type.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_ICACHE_REGION_0
  – LL_ICACHE_REGION_1
  – LL_ICACHE_REGION_2
  – LL_ICACHE_REGION_3

Return values

- **Returned:** value can be one of the following values:
  – LL_ICACHE_OUTPUT_BURST_WRAP
  – LL_ICACHE_OUTPUT_BURST_INCR

Reference Manual to LL API cross reference:

- CRRx HBURST LL_ICACHE_GetRegionOutputBurstType

LL_ICACHE_SetRegionMasterPort

Function name

__STATIC_INLINE void LL_ICACHE_SetRegionMasterPort (uint32_t Region, uint32_t Port)

Function description

Select the memory remapped region cache master port.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_ICACHE_REGION_0
  – LL_ICACHE_REGION_1
  – LL_ICACHE_REGION_2
  – LL_ICACHE_REGION_3
- **Port:** This parameter can be one of the following values:
  – LL_ICACHE_MASTER1_PORT
  – LL_ICACHE_MASTER2_PORT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRRx MSTSEL LL_ICACHE_SetRegionMasterPort

LL_ICACHE_GetRegionMasterPort

Function name

__STATIC_INLINE uint32_t LL_ICACHE_GetRegionMasterPort (uint32_t Region)

Function description

Get the selected the memory remapped region cache master port.

Parameters

- **Region:** This parameter can be one of the following values:
  – LL_ICACHE_REGION_0
  – LL_ICACHE_REGION_1
  – LL_ICACHE_REGION_2
  – LL_ICACHE_REGION_3

Return values

- **Returned:** value can be one of the following values:
  – LL_ICACHE_MASTER1_PORT
  – LL_ICACHE_MASTER2_PORT

Reference Manual to LL API cross reference:

- CRRx MSTSEL LL_ICACHE_GetRegionMasterPort

LL_ICACHE_ConfigRegion

Function name

void LL_ICACHE_ConfigRegion (uint32_t Region, const LL_ICACHE_RegionTypeDef *const pICACHE_RegionStruct)

Function description

Configure and enable the memory remapped region.

Parameters

- **Region:** This parameter can be one of the following values:
  - LL_ICACHE_REGION_0
  - LL_ICACHE_REGION_1
  - LL_ICACHE_REGION_2
  - LL_ICACHE_REGION_3
- **pICACHE_RegionStruct:** pointer to a LL_ICACHE_RegionTypeDef structure.

Return values

- **None:**

Notes

- The Instruction Cache and corresponding region must be disabled.

106.3 ICACHE Firmware driver defines

The following section lists the various define and macros of the module.

106.3.1 ICACHE

ICACHE

Clear Flags Defines

LL_ICACHE_FCR_CBSYENDF

Busy end flag

LL_ICACHE_FCR_CERRF

Cache error flag

Get Flags Defines

LL_ICACHE_SR_BUSYF

Busy flag

LL_ICACHE_SR_BSYENDF

Busy end flag

LL_ICACHE_SR_ERRF

Cache error flag

IT Defines

LL_ICACHE_IER_BSYENDIE

Busy end interrupt

LL_ICACHE_IER_ERRIE

Cache error interrupt

Monitor type

LL_ICACHE_MONITOR_HIT

Hit monitor counter

LL_ICACHE_MONITOR_MISS

Miss monitor counter

LL_ICACHE_MONITOR_ALL

All monitors counters

Remapped Output burst type

LL_ICACHE_OUTPUT_BURST_WRAP

WRAP

LL_ICACHE_OUTPUT_BURST_INCR

INCR

Remapped Region number

LL_ICACHE_REGION_0

Region 0

LL_ICACHE_REGION_1

Region 1

LL_ICACHE_REGION_2

Region 2

LL_ICACHE_REGION_3

Region 3

Remapped Region size

LL_ICACHE_REGIONSIZE_2MB

Region size 2MB

LL_ICACHE_REGIONSIZE_4MB

Region size 4MB

LL_ICACHE_REGIONSIZE_8MB

Region size 8MB

LL_ICACHE_REGIONSIZE_16MB

Region size 16MB

LL_ICACHE_REGIONSIZE_32MB

Region size 32MB

LL_ICACHE_REGIONSIZE_64MB

Region size 64MB

LL_ICACHE_REGIONSIZE_128MB

Region size 128MB

Remapped Traffic route

LL_ICACHE_MASTER1_PORT

Master1 port

LL_ICACHE_MASTER2_PORT

Master2 port

Ways selection

LL_ICACHE_1WAY

1-way cache (direct mapped cache)

LL_ICACHE_2WAYS

2-ways set associative cache (default)

Common write and read registers Macros

LL_ICACHE_WriteReg

Description:

- Write a value in ICACHE register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_ICACHE_ReadReg

Description:

- Read a value in ICACHE register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

107 LL IWDG Generic Driver

107.1 IWDG Firmware driver API description

The following section lists the various functions of the IWDG library.

107.1.1 Detailed description of functions

LL_IWDG_Enable

Function name

__STATIC_INLINE void LL_IWDG_Enable (IWDG_TypeDef * IWDGx)

Function description

Start the Independent Watchdog.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Notes

- Except if the hardware watchdog option is selected

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_Enable

LL_IWDG_ReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_ReloadCounter (IWDG_TypeDef * IWDGx)

Function description

Reloads IWDG counter with value defined in the reload register.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_ReloadCounter

LL_IWDG_EnableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_EnableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Enable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_EnableWriteAccess

LL_IWDG_DisableWriteAccess

Function name

__STATIC_INLINE void LL_IWDG_DisableWriteAccess (IWDG_TypeDef * IWDGx)

Function description

Disable write access to IWDG_PR, IWDG_RLR and IWDG_WINR registers.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- KR KEY LL_IWDG_DisableWriteAccess

LL_IWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_IWDG_SetPrescaler (IWDG_TypeDef * IWDGx, uint32_t Prescaler)

Function description

Select the prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  – LL_IWDG_PRESCALER_4
  – LL_IWDG_PRESCALER_8
  – LL_IWDG_PRESCALER_16
  – LL_IWDG_PRESCALER_32
  – LL_IWDG_PRESCALER_64
  – LL_IWDG_PRESCALER_128
  – LL_IWDG_PRESCALER_256
  – LL_IWDG_PRESCALER_512
  – LL_IWDG_PRESCALER_1024

Return values

- **None:**

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_SetPrescaler

LL_IWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetPrescaler (const IWDG_TypeDef * IWDGx)

Function description

Get the selected prescaler of the IWDG.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_IWDG_PRESCALER_4
  - LL_IWDG_PRESCALER_8
  - LL_IWDG_PRESCALER_16
  - LL_IWDG_PRESCALER_32
  - LL_IWDG_PRESCALER_64
  - LL_IWDG_PRESCALER_128
  - LL_IWDG_PRESCALER_256
  - LL_IWDG_PRESCALER_512
  - LL_IWDG_PRESCALER_1024

Reference Manual to LL API cross reference:

- PR PR LL_IWDG_GetPrescaler

LL_IWDG_SetReloadCounter

Function name

__STATIC_INLINE void LL_IWDG_SetReloadCounter (IWDG_TypeDef * IWDGx, uint32_t Counter)

Function description

Specify the IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance
- **Counter:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_SetReloadCounter

LL_IWDG_GetReloadCounter

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetReloadCounter (const IWDG_TypeDef * IWDGx)

Function description

Get the specified IWDG down-counter reload value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- RLR RL LL_IWDG_GetReloadCounter

LL_IWDG_SetWindow

Function name

__STATIC_INLINE void LL_IWDG_SetWindow (IWDG_TypeDef * IWDGx, uint32_t Window)

Function description

Specify high limit of the window value to be compared to the down-counter.

Parameters

- **IWDGx:** IWDG Instance
- **Window:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_SetWindow

LL_IWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetWindow (const IWDG_TypeDef * IWDGx)

Function description

Get the high limit of the window value specified.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- WINR WIN LL_IWDG_GetWindow

LL_IWDG_SetEwiTime

Function name

__STATIC_INLINE void LL_IWDG_SetEwiTime (IWDG_TypeDef * IWDGx, uint32_t Time)

Function description

Specify comparator value that will be used to trig Early Wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance
- **Time:** Value between Min_Data=0 and Max_Data=0x0FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIT LL_IWDG_SetEwiTime

LL_IWDG_GetEwiTime

Function name

__STATIC_INLINE uint32_t LL_IWDG_GetEwiTime (const IWDG_TypeDef * IWDGx)

Function description

Get the Early Wakeup interrupt comparator value.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **Value:** between Min_Data=0 and Max_Data=0x0FFF

Reference Manual to LL API cross reference:

- EWCR EWIT LL_IWDG_GetEwiTime

LL_IWDG_EnableIT_EWI

Function name

__STATIC_INLINE void LL_IWDG_EnableIT_EWI (IWDG_TypeDef * IWDGx)

Function description

Enable Early wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_EnableIT_EWI

LL_IWDG_DisableIT_EWI

Function name

__STATIC_INLINE void LL_IWDG_DisableIT_EWI (IWDG_TypeDef * IWDGx)

Function description

Disable Early wakeup interrupt.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_DisableIT_EWI

LL_IWDG_IsEnabledIT_EWI

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsEnabledIT_EWI (const IWDG_TypeDef * IWDGx)

Function description

Indicates whether Early wakeup interrupt is enable.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIE LL_IWDG_IsEnabledIT_EWI

LL_IWDG_IsActiveFlag_PVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_PVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Prescaler Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsActiveFlag_PVU

LL_IWDG_IsActiveFlag_RVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_RVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Reload Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RVU LL_IWDG_IsActiveFlag_RVU

LL_IWDG_IsActiveFlag_WVU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_WVU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag Window Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR WVU LL_IWDG_IsActiveFlag_WVU

LL_IWDG_IsActiveFlag_EWU

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_EWU (const IWDG_TypeDef * IWDGx)

Function description

Check if flag EWI Value Update is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EVU LL_IWDG_IsActiveFlag_EWU

LL_IWDG_IsReady

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsReady (const IWDG_TypeDef * IWDGx)

Function description

Check if all flags Prescaler, Reload, Window & Early Interrupt Value Update are reset or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bits (1 or 0).

Reference Manual to LL API cross reference:

- SR PVU LL_IWDG_IsReady
- SR RVU LL_IWDG_IsReady
- SR WVU LL_IWDG_IsReady
- SR EWU LL_IWDG_IsReady

LL_IWDG_IsActiveFlag_EWIF

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsActiveFlag_EWIF (const IWDG_TypeDef * IWDGx)

Function description

Check if Early Wakeup interrupt flag is set or not.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EWIF LL_IWDG_IsActiveFlag_EWIF

LL_IWDG_ClearFlag_EWIF

Function name

__STATIC_INLINE void LL_IWDG_ClearFlag_EWIF (IWDG_TypeDef * IWDGx)

Function description

Clear the Early Wakeup interrupt flag.

Parameters

- **IWDGx:** IWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EWCR EWIC LL_IWDG_ClearFlag_EWIF

107.2 IWDG Firmware driver defines

The following section lists the various define and macros of the module.

107.2.1 IWDG

IWDG

Get Flags Defines

LL_IWDG_SR_PVU

Watchdog prescaler value update

LL_IWDG_SR_RVU

Watchdog counter reload value update

LL_IWDG_SR_WVU

Watchdog counter window value update

Prescaler Divider

LL_IWDG_PRESCALER_4

Divider by 4

LL_IWDG_PRESCALER_8

Divider by 8

LL_IWDG_PRESCALER_16

Divider by 16

LL_IWDG_PRESCALER_32

Divider by 32

LL_IWDG_PRESCALER_64

Divider by 64

LL_IWDG_PRESCALER_128

Divider by 128

LL_IWDG_PRESCALER_256

Divider by 256

LL_IWDG_PRESCALER_512

Divider by 512

LL_IWDG_PRESCALER_1024

Divider by 1024

Common Write and read registers Macros

LL_IWDG_WriteReg

Description:

- Write a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_IWDG_ReadReg

Description:

- Read a value in IWDG register.

Parameters:

- __INSTANCE__: IWDG Instance
- __REG__: Register to be read

Return value:

- Register: value

108 LL LPGPIO Generic Driver

108.1 LPGPIO Firmware driver registers structures

108.1.1 LL_LPGPIO_InitTypeDef

LL_LPGPIO_InitTypeDef is defined in the stm32u5xx_ll_lpgpio.h

Data Fields

- ***uint32_t Pin***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t LL_LPGPIO_InitTypeDef::Pin***
  Specifies the LPGPIO pins to be configured. This parameter can be any value of ***LPGPIO_LL_EC_PIN***
- ***uint32_t LL_LPGPIO_InitTypeDef::Mode***
  Specifies the operating mode for the selected pins. This parameter can be a value of ***LPGPIO_LL_EC_MODE***.

108.2 LPGPIO Firmware driver API description

The following section lists the various functions of the LPGPIO library.

108.2.1 Detailed description of functions

LL_LPGPIO_SetPinMode

Function name

__STATIC_INLINE void LL_LPGPIO_SetPinMode (GPIO_TypeDef * LPGPIOx, uint32_t Pin, uint32_t Mode)

Function description

Configure lpgpio mode for a dedicated pin on dedicated port.

Parameters

- **LPGPIOx:** LPGPIO Port
- **Pin:** This parameter can be one of the following values:
  - LL_LPGPIO_PIN_0
  - LL_LPGPIO_PIN_1
  - LL_LPGPIO_PIN_2
  - LL_LPGPIO_PIN_3
  - LL_LPGPIO_PIN_4
  - LL_LPGPIO_PIN_5
  - LL_LPGPIO_PIN_6
  - LL_LPGPIO_PIN_7
  - LL_LPGPIO_PIN_8
  - LL_LPGPIO_PIN_9
  - LL_LPGPIO_PIN_10
  - LL_LPGPIO_PIN_11
  - LL_LPGPIO_PIN_12
  - LL_LPGPIO_PIN_13
  - LL_LPGPIO_PIN_14
  - LL_LPGPIO_PIN_15
- **Mode:** This parameter can be one of the following values:
  - LL_LPGPIO_MODE_INPUT
  - LL_LPGPIO_MODE_OUTPUT

Return values

- **None:**

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_LPGPIO_SetPinMode

LL_LPGPIO_GetPinMode

Function name

__STATIC_INLINE uint32_t LL_LPGPIO_GetPinMode (const GPIO_TypeDef * LPGPIOx, uint32_t Pin)

Function description

Return lpgpio mode for a dedicated pin on dedicated port.

Parameters

- **LPGPIOx:** LPGPIO Port
- **Pin:** This parameter can be one of the following values:
  – LL_LPGPIO_PIN_0
  – LL_LPGPIO_PIN_1
  – LL_LPGPIO_PIN_2
  – LL_LPGPIO_PIN_3
  – LL_LPGPIO_PIN_4
  – LL_LPGPIO_PIN_5
  – LL_LPGPIO_PIN_6
  – LL_LPGPIO_PIN_7
  – LL_LPGPIO_PIN_8
  – LL_LPGPIO_PIN_9
  – LL_LPGPIO_PIN_10
  – LL_LPGPIO_PIN_11
  – LL_LPGPIO_PIN_12
  – LL_LPGPIO_PIN_13
  – LL_LPGPIO_PIN_14
  – LL_LPGPIO_PIN_15

Return values

- **Returned:** value can be one of the following values:
  – LL_LPGPIO_MODE_INPUT
  – LL_LPGPIO_MODE_OUTPUT

Notes

- I/O mode can be Input mode, General purpose output, Alternate function mode or Analog.
- Warning: only one pin can be passed as parameter.

Reference Manual to LL API cross reference:

- MODER MODEy LL_LPGPIO_GetPinMode

LL_LPGPIO_ReadInputPort

Function name

__STATIC_INLINE uint32_t LL_LPGPIO_ReadInputPort (const GPIO_TypeDef * LPGPIOx)

Function description

Return full input data register value for a dedicated port.

Parameters

- **LPGPIOx:** LPGPIO Port

Return values

- **Input:** data register value of port

Reference Manual to LL API cross reference:

- IDR IDy LL_LPGPIO_ReadInputPort

LL_LPGPIO_IsInputPinSet

Function name

__STATIC_INLINE uint32_t LL_LPGPIO_IsInputPinSet (const GPIO_TypeDef * LPGPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **LPGPIOx:** LPGPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_LPGPIO_PIN_0
  - LL_LPGPIO_PIN_1
  - LL_LPGPIO_PIN_2
  - LL_LPGPIO_PIN_3
  - LL_LPGPIO_PIN_4
  - LL_LPGPIO_PIN_5
  - LL_LPGPIO_PIN_6
  - LL_LPGPIO_PIN_7
  - LL_LPGPIO_PIN_8
  - LL_LPGPIO_PIN_9
  - LL_LPGPIO_PIN_10
  - LL_LPGPIO_PIN_11
  - LL_LPGPIO_PIN_12
  - LL_LPGPIO_PIN_13
  - LL_LPGPIO_PIN_14
  - LL_LPGPIO_PIN_15
  - LL_LPGPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IDR IDy LL_LPGPIO_IsInputPinSet

LL_LPGPIO_WriteOutputPort

Function name

__STATIC_INLINE void LL_LPGPIO_WriteOutputPort (GPIO_TypeDef * LPGPIOx, uint32_t PortValue)

Function description

Write output data register for the port.

Parameters

- **LPGPIOx:** LPGPIO Port
- **PortValue:** Level value for each pin of the port Value between 0 and 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_LPGPIO_WriteOutputPort

LL_LPGPIO_ReadOutputPort

Function name

__STATIC_INLINE uint32_t LL_LPGPIO_ReadOutputPort (const GPIO_TypeDef * LPGPIOx)

Function description

Return full output data register value for a dedicated port.

Parameters

- **LPGPIOx:** LPGPIO Port

Return values

- **Output:** data register value of port

Reference Manual to LL API cross reference:

- ODR ODy LL_LPGPIO_ReadOutputPort

LL_LPGPIO_IsOutputPinSet

Function name

__STATIC_INLINE uint32_t LL_LPGPIO_IsOutputPinSet (const GPIO_TypeDef * LPGPIOx, uint32_t PinMask)

Function description

Return if input data level for several pins of dedicated port is high or low.

Parameters

- **LPGPIOx:** LPGPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_LPGPIO_PIN_0
  – LL_LPGPIO_PIN_1
  – LL_LPGPIO_PIN_2
  – LL_LPGPIO_PIN_3
  – LL_LPGPIO_PIN_4
  – LL_LPGPIO_PIN_5
  – LL_LPGPIO_PIN_6
  – LL_LPGPIO_PIN_7
  – LL_LPGPIO_PIN_8
  – LL_LPGPIO_PIN_9
  – LL_LPGPIO_PIN_10
  – LL_LPGPIO_PIN_11
  – LL_LPGPIO_PIN_12
  – LL_LPGPIO_PIN_13
  – LL_LPGPIO_PIN_14
  – LL_LPGPIO_PIN_15
  – LL_LPGPIO_PIN_ALL

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ODR ODy LL_LPGPIO_IsOutputPinSet

LL_LPGPIO_SetOutputPin

Function name

__STATIC_INLINE void LL_LPGPIO_SetOutputPin (GPIO_TypeDef * LPGPIOx, uint32_t PinMask)

Function description

Set several pins to high level on dedicated gpio port.

Parameters

- **LPGPIOx:** LPGPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_LPGPIO_PIN_0
  - LL_LPGPIO_PIN_1
  - LL_LPGPIO_PIN_2
  - LL_LPGPIO_PIN_3
  - LL_LPGPIO_PIN_4
  - LL_LPGPIO_PIN_5
  - LL_LPGPIO_PIN_6
  - LL_LPGPIO_PIN_7
  - LL_LPGPIO_PIN_8
  - LL_LPGPIO_PIN_9
  - LL_LPGPIO_PIN_10
  - LL_LPGPIO_PIN_11
  - LL_LPGPIO_PIN_12
  - LL_LPGPIO_PIN_13
  - LL_LPGPIO_PIN_14
  - LL_LPGPIO_PIN_15
  - LL_LPGPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BSRR BSy LL_LPGPIO_SetOutputPin

LL_LPGPIO_ResetOutputPin

Function name

__STATIC_INLINE void LL_LPGPIO_ResetOutputPin (GPIO_TypeDef * LPGPIOx, uint32_t PinMask)

Function description

Set several pins to low level on dedicated gpio port.

Parameters

- **LPGPIOx:** LPGPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  – LL_LPGPIO_PIN_0
  – LL_LPGPIO_PIN_1
  – LL_LPGPIO_PIN_2
  – LL_LPGPIO_PIN_3
  – LL_LPGPIO_PIN_4
  – LL_LPGPIO_PIN_5
  – LL_LPGPIO_PIN_6
  – LL_LPGPIO_PIN_7
  – LL_LPGPIO_PIN_8
  – LL_LPGPIO_PIN_9
  – LL_LPGPIO_PIN_10
  – LL_LPGPIO_PIN_11
  – LL_LPGPIO_PIN_12
  – LL_LPGPIO_PIN_13
  – LL_LPGPIO_PIN_14
  – LL_LPGPIO_PIN_15
  – LL_LPGPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- BRR BRy LL_LPGPIO_ResetOutputPin

LL_LPGPIO_TogglePin

Function name

__STATIC_INLINE void LL_LPGPIO_TogglePin (GPIO_TypeDef * LPGPIOx, uint32_t PinMask)

Function description

Toggle data value for several pin of dedicated port.

Parameters

- **LPGPIOx:** LPGPIO Port
- **PinMask:** This parameter can be a combination of the following values:
  - LL_LPGPIO_PIN_0
  - LL_LPGPIO_PIN_1
  - LL_LPGPIO_PIN_2
  - LL_LPGPIO_PIN_3
  - LL_LPGPIO_PIN_4
  - LL_LPGPIO_PIN_5
  - LL_LPGPIO_PIN_6
  - LL_LPGPIO_PIN_7
  - LL_LPGPIO_PIN_8
  - LL_LPGPIO_PIN_9
  - LL_LPGPIO_PIN_10
  - LL_LPGPIO_PIN_11
  - LL_LPGPIO_PIN_12
  - LL_LPGPIO_PIN_13
  - LL_LPGPIO_PIN_14
  - LL_LPGPIO_PIN_15
  - LL_LPGPIO_PIN_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- ODR ODy LL_LPGPIO_TogglePin

LL_LPGPIO_DeInit

Function name

ErrorStatus LL_LPGPIO_DeInit (const GPIO_TypeDef * LPGPIOx)

Function description

De-initialize LPGPIO registers (Registers restored to their default values).

Parameters

- **LPGPIOx:** LPGPIO Port

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPGPIO registers are de-initialized
  - ERROR: Wrong LPGPIO Port

LL_LPGPIO_Init

Function name

ErrorStatus LL_LPGPIO_Init (GPIO_TypeDef * LPGPIOx, const LL_LPGPIO_InitTypeDef *const LPGPIO_InitStruct)

Function description

Initialize LPGPIO registers according to the specified parameters in LPGPIO_InitStruct.

Parameters

- **LPGPIOx:** LPGPIO Port
- **LPGPIO_InitStruct:** pointer to a LL_LPGPIO_InitTypeDef structure that contains the configuration information for the specified LPGPIO peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPGPIO registers are initialized according to LPGPIO_InitStruct content
  - ERROR: Not applicable

LL_LPGPIO_StructInit

Function name

void LL_LPGPIO_StructInit (LL_LPGPIO_InitTypeDef * LPGPIO_InitStruct)

Function description

Set each LL_LPGPIO_InitTypeDef field to default value.

Parameters

- **LPGPIO_InitStruct:** pointer to a LL_LPGPIO_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

108.3 LPGPIO Firmware driver defines

The following section lists the various define and macros of the module.

108.3.1 LPGPIO

LPGPIO

Mode

LL_LPGPIO_MODE_INPUT

Select input mode

LL_LPGPIO_MODE_OUTPUT

Select output mode

PIN

LL_LPGPIO_PIN_0

Select pin 0

LL_LPGPIO_PIN_1

Select pin 1

LL_LPGPIO_PIN_2

Select pin 2

LL_LPGPIO_PIN_3

Select pin 3

LL_LPGPIO_PIN_4

Select pin 4

LL_LPGPIO_PIN_5

Select pin 5

LL_LPGPIO_PIN_6

Select pin 6

LL_LPGPIO_PIN_7

Select pin 7

LL_LPGPIO_PIN_8

Select pin 8

LL_LPGPIO_PIN_9

Select pin 9

LL_LPGPIO_PIN_10

Select pin 10

LL_LPGPIO_PIN_11

Select pin 11

LL_LPGPIO_PIN_12

Select pin 12

LL_LPGPIO_PIN_13

Select pin 13

LL_LPGPIO_PIN_14

Select pin 14

LL_LPGPIO_PIN_15

Select pin 15

LL_LPGPIO_PIN_ALL

Select all pins

Common Write and read registers Macros

LL_LPGPIO_WriteReg

Description:

- Write a value in LPGPIO register.

Parameters:

- __INSTANCE__: LPGPIO Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPGPIO_ReadReg

Description:

- Read a value in LPGPIO register.

Parameters:

- __INSTANCE__: LPGPIO Instance
- __REG__: Register to be read

Return value:

- Register: value

109 LL LPTIM Generic Driver

109.1 LPTIM Firmware driver registers structures

109.1.1 LL_LPTIM_InitTypeDef

LL_LPTIM_InitTypeDef is defined in the stm32u5xx_ll_lptim.h

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t Prescaler***
- ***uint32_t Waveform***

Field Documentation

- ***uint32_t LL_LPTIM_InitTypeDef::ClockSource***
  Specifies the source of the clock used by the LPTIM instance. This parameter can be a value of *LPTIM_LL_EC_CLK_SOURCE*.This feature can be modified afterwards using unitary function **LL_LPTIM_SetClockSource()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Prescaler***
  Specifies the prescaler division ratio. This parameter can be a value of *LPTIM_LL_EC_PRESCALER*.This feature can be modified afterwards using using unitary function **LL_LPTIM_SetPrescaler()**.
- ***uint32_t LL_LPTIM_InitTypeDef::Waveform***
  Specifies the waveform shape. This parameter can be a value of *LPTIM_LL_EC_OUTPUT_WAVEFORM*.This feature can be modified afterwards using unitary function **LL_LPTIM_SetWaveform()**.

109.2 LPTIM Firmware driver API description

The following section lists the various functions of the LPTIM library.

109.2.1 Detailed description of functions

LL_LPTIM_DeInit

Function name

ErrorStatus LL_LPTIM_DeInit (const LPTIM_TypeDef * LPTIMx)

Function description

Set LPTIMx registers to their reset values.

Parameters

- **LPTIMx:** LP Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: LPTIMx registers are de-initialized
  – ERROR: invalid LPTIMx instance

LL_LPTIM_StructInit

Function name

void LL_LPTIM_StructInit (LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Set each fields of the LPTIM_InitStruct structure to its default value.

Parameters

- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **None:**

LL_LPTIM_Init

Function name

ErrorStatus LL_LPTIM_Init (LPTIM_TypeDef * LPTIMx, const LL_LPTIM_InitTypeDef * LPTIM_InitStruct)

Function description

Configure the LPTIMx peripheral according to the specified parameters.

Parameters

- **LPTIMx:** LP Timer Instance
- **LPTIM_InitStruct:** pointer to a LL_LPTIM_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: LPTIMx instance has been initialized
  – ERROR: LPTIMx instance hasn't been initialized

Notes

- LL_LPTIM_Init can only be called when the LPTIM instance is disabled.
- LPTIMx can be disabled using unitary function LL_LPTIM_Disable().

LL_LPTIM_Enable

Function name

__STATIC_INLINE void LL_LPTIM_Enable (LPTIM_TypeDef * LPTIMx)

Function description

Enable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After setting the ENABLE bit, a delay of two counter clock is needed before the LPTIM instance is actually enabled.

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Enable

LL_LPTIM_Disable

Function name

__STATIC_INLINE void LL_LPTIM_Disable (LPTIM_TypeDef * LPTIMx)

Function description

Disable the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_Disable

LL_LPTIM_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabled (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM instance is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ENABLE LL_LPTIM_IsEnabled

LL_LPTIM_StartCounter

Function name

__STATIC_INLINE void LL_LPTIM_StartCounter (LPTIM_TypeDef * LPTIMx, uint32_t OperatingMode)

Function description

Starts the LPTIM counter in the desired mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **OperatingMode:** This parameter can be one of the following values:
  - LL_LPTIM_OPERATING_MODE_CONTINUOUS
  - LL_LPTIM_OPERATING_MODE_ONESHOT

Return values

- **None:**

Notes

- LPTIM instance must be enabled before starting the counter.
- It is possible to change on the fly from One Shot mode to Continuous mode.

Reference Manual to LL API cross reference:

- CR CNTSTRT LL_LPTIM_StartCounter
- CR SNGSTRT LL_LPTIM_StartCounter

LL_LPTIM_EnableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_EnableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Enable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- After calling this function any read access to LPTIM_CNT register will asynchronously reset the LPTIM_CNT register content.

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_EnableResetAfterRead

LL_LPTIM_DisableResetAfterRead

Function name

__STATIC_INLINE void LL_LPTIM_DisableResetAfterRead (LPTIM_TypeDef * LPTIMx)

Function description

Disable reset after read.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_DisableResetAfterRead

LL_LPTIM_IsEnabledResetAfterRead

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledResetAfterRead (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the reset after read feature is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RSTARE LL_LPTIM_IsEnabledResetAfterRead

LL_LPTIM_ResetCounter

Function name

__STATIC_INLINE void LL_LPTIM_ResetCounter (LPTIM_TypeDef * LPTIMx)

Function description

Reset of the LPTIM_CNT counter register (synchronous).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- Due to the synchronous nature of this reset, it only takes place after a synchronization delay of 3 LPTIM core clock cycles (LPTIM core clock may be different from APB clock).
- COUNTRST is automatically cleared by hardware

Reference Manual to LL API cross reference:

- CR COUNTRST LL_LPTIM_ResetCounter
-

LL_LPTIM_SetUpdateMode

Function name

__STATIC_INLINE void LL_LPTIM_SetUpdateMode (LPTIM_TypeDef * LPTIMx, uint32_t UpdateMode)

Function description

Set the LPTIM registers update mode (enable/disable register preload)

Parameters

- **LPTIMx:** Low-Power Timer instance
- **UpdateMode:** This parameter can be one of the following values:
  – LL_LPTIM_UPDATE_MODE_IMMEDIATE
  – LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_SetUpdateMode

LL_LPTIM_GetUpdateMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetUpdateMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the LPTIM registers update mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_UPDATE_MODE_IMMEDIATE
  – LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

Reference Manual to LL API cross reference:

- CFGR PRELOAD LL_LPTIM_GetUpdateMode

LL_LPTIM_SetAutoReload

Function name

__STATIC_INLINE void LL_LPTIM_SetAutoReload (LPTIM_TypeDef * LPTIMx, uint32_t AutoReload)

Function description

Set the auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- The LPTIMx_ARR register content must only be modified when the LPTIM is enabled
- After a write to the LPTIMx_ARR register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the ARROK flag is set, will lead to unpredictable results.
- autoreload value be strictly greater than the compare value.

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_SetAutoReload

LL_LPTIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetAutoReload (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual auto reload value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **AutoReload:** Value between Min_Data=0x0001 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- ARR ARR LL_LPTIM_GetAutoReload

LL_LPTIM_SetRepetition

Function name

__STATIC_INLINE void LL_LPTIM_SetRepetition (LPTIM_TypeDef * LPTIMx, uint32_t Repetition)

Function description

Set the repetition value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Repetition:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- The LPTIMx_RCR register content must only be modified when the LPTIM is enabled

Reference Manual to LL API cross reference:

- RCR REP LL_LPTIM_SetRepetition

LL_LPTIM_GetRepetition

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetRepetition (const LPTIM_TypeDef * LPTIMx)

Function description

Get the repetition value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Repetition:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RCR REP LL_LPTIM_GetRepetition

LL_LPTIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_LPTIM_CC_EnableChannel (LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Enable capture/compare channel.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_EnableChannel
- CCMR1 CC2E LL_LPTIM_CC_EnableChannel

LL_LPTIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_LPTIM_CC_DisableChannel (LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Disable capture/compare channel.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_DisableChannel
- CCMR1 CC2E LL_LPTIM_CC_DisableChannel

LL_LPTIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_LPTIM_CC_IsEnabledChannel (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Indicate whether channel is enabled.

Parameters

- **LPTIMx:** LPTimer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 CC1E LL_LPTIM_CC_IsEnabledChannel
- CCMR1 CC2E LL_LPTIM_CC_IsEnabledChannel

LL_LPTIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetCompareCH1 (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CCR1 register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMP1OK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_OC_SetCompareCH1

LL_LPTIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetCompareCH1 (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_OC_GetCompareCH1

LL_LPTIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetCompareCH2 (LPTIM_TypeDef * LPTIMx, uint32_t CompareValue)

Function description

Set the compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- After a write to the LPTIMx_CCR2 register a new write operation to the same register can only be performed when the previous write operation is completed. Any successive write before the CMP2OK flag is set, will lead to unpredictable results.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_OC_SetCompareCH2

LL_LPTIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetCompareCH2 (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual compare value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CompareValue:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_OC_GetCompareCH2

LL_LPTIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual counter value.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Counter:** value

Notes

- When the LPTIM instance is running with an asynchronous clock, reading the LPTIMx_CNT register may return unreliable values. So in this case it is necessary to perform two consecutive read accesses and verify that the two returned values are identical.

Reference Manual to LL API cross reference:

- CNT CNT LL_LPTIM_GetCounter

LL_LPTIM_SetCounterMode

Function name

__STATIC_INLINE void LL_LPTIM_SetCounterMode (LPTIM_TypeDef * LPTIMx, uint32_t CounterMode)

Function description

Set the counter mode (selection of the LPTIM counter clock source).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **CounterMode:** This parameter can be one of the following values:
  – LL_LPTIM_COUNTER_MODE_INTERNAL
  – LL_LPTIM_COUNTER_MODE_EXTERNAL

Return values

- **None:**

Notes

- The counter mode can be set only when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_SetCounterMode

LL_LPTIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetCounterMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get the counter mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_COUNTER_MODE_INTERNAL
  – LL_LPTIM_COUNTER_MODE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR COUNTMODE LL_LPTIM_GetCounterMode

LL_LPTIM_SetWaveform

Function name

__STATIC_INLINE void LL_LPTIM_SetWaveform (LPTIM_TypeDef * LPTIMx, uint32_t Waveform)

Function description

Set waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Waveform:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_SetWaveform

LL_LPTIM_GetWaveform

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetWaveform (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual waveform shape.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_WAVEFORM_PWM
  - LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

Reference Manual to LL API cross reference:

- CFGR WAVE LL_LPTIM_GetWaveform

LL_LPTIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_OC_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_OC_SetPolarity
-
- CCMR1 CC2P LL_LPTIM_OC_SetPolarity
-

LL_LPTIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_OC_GetPolarity (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_LPTIM_CHANNEL_CH1
  - LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_OUTPUT_POLARITY_REGULAR
  - LL_LPTIM_OUTPUT_POLARITY_INVERSE

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_OC_GetPolarity
-
- CCMR1 CC2P LL_LPTIM_OC_GetPolarity
-

LL_LPTIM_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Prescaler)

Function description

Set actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Prescaler:** This parameter can be one of the following values:
  – LL_LPTIM_PRESCALER_DIV1
  – LL_LPTIM_PRESCALER_DIV2
  – LL_LPTIM_PRESCALER_DIV4
  – LL_LPTIM_PRESCALER_DIV8
  – LL_LPTIM_PRESCALER_DIV16
  – LL_LPTIM_PRESCALER_DIV32
  – LL_LPTIM_PRESCALER_DIV64
  – LL_LPTIM_PRESCALER_DIV128

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When the LPTIM is configured to be clocked by an internal clock source and the LPTIM counter is configured to be updated by active edges detected on the LPTIM external Input1, the internal clock provided to the LPTIM must be not be prescaled.

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_SetPrescaler

LL_LPTIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetPrescaler (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual prescaler division ratio.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_PRESCALER_DIV1
  – LL_LPTIM_PRESCALER_DIV2
  – LL_LPTIM_PRESCALER_DIV4
  – LL_LPTIM_PRESCALER_DIV8
  – LL_LPTIM_PRESCALER_DIV16
  – LL_LPTIM_PRESCALER_DIV32
  – LL_LPTIM_PRESCALER_DIV64
  – LL_LPTIM_PRESCALER_DIV128

Reference Manual to LL API cross reference:

- CFGR PRESC LL_LPTIM_GetPrescaler

LL_LPTIM_SetInput1Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput1Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 1 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  – LL_LPTIM_INPUT1_SRC_GPIO
  – LL_LPTIM_INPUT1_SRC_COMP1
  – LL_LPTIM_INPUT1_SRC_COMP2 (*) (*) Value not defined for all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN1SEL LL_LPTIM_SetInput1Src

LL_LPTIM_SetInput2Src

Function name

__STATIC_INLINE void LL_LPTIM_SetInput2Src (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input 2 source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  – LL_LPTIM_INPUT2_SRC_GPIO
  – LL_LPTIM_INPUT2_SRC_COMP2 (*) (*) Value not defined for all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IN2SEL LL_LPTIM_SetInput2Src

LL_LPTIM_SetRemap

Function name

__STATIC_INLINE void LL_LPTIM_SetRemap (LPTIM_TypeDef * LPTIMx, uint32_t Src)

Function description

Set LPTIM input source (default GPIO).

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Src:** This parameter can be one of the following values:
  – LL_LPTIM_LPTIM1_IC1_RMP_GPIO
  – LL_LPTIM_LPTIM1_IC1_RMP_COMP1
  – LL_LPTIM_LPTIM1_IC1_RMP_COMP2 (*)
  – LL_LPTIM_LPTIM1_IC2_RMP_GPIO
  – LL_LPTIM_LPTIM1_IC2_RMP_LSI
  – LL_LPTIM_LPTIM1_IC2_RMP_LSE
  – LL_LPTIM_LPTIM2_IC1_RMP_GPIO
  – LL_LPTIM_LPTIM2_IC1_RMP_COMP1
  – LL_LPTIM_LPTIM2_IC1_RMP_COMP2 (*)
  – LL_LPTIM_LPTIM2_IC2_RMP_GPIO
  – LL_LPTIM_LPTIM2_IC2_RMP_HSI
  – LL_LPTIM_LPTIM2_IC2_RMP_MSIS_1024
  – LL_LPTIM_LPTIM2_IC2_RMP_MSIS_4
  – LL_LPTIM_LPTIM3_IC1_RMP_GPIO
  – LL_LPTIM_LPTIM3_IC1_RMP_COMP1
  – LL_LPTIM_LPTIM3_IC1_RMP_COMP2 (*)

  (*) Value not defined in all devices.
-

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 IC1SEL LL_LPTIM_SetRemap
- CFGR2 IC2SEL LL_LPTIM_SetRemap

LL_LPTIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetPolarity (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of IC channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2
- **Polarity:** This parameter can be one of the following values:
  – LL_LPTIM_ICPOLARITY_RISING
  – LL_LPTIM_ICPOLARITY_FALLING
  – LL_LPTIM_ICPOLARITY_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_IC_SetPolarity
-
- CCMR1 CC2P LL_LPTIM_IC_SetPolarity
-

LL_LPTIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetPolarity (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the polarity of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_ICPOLARITY_RISING
  – LL_LPTIM_ICPOLARITY_FALLING
  – LL_LPTIM_ICPOLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CCMR1 CC1P LL_LPTIM_IC_GetPolarity
-
- CCMR1 CC2P LL_LPTIM_IC_GetPolarity
-

LL_LPTIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetFilter (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Filter)

Function description

Set the filter of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2
- **Filter:** This parameter can be one of the following values:
  – LL_LPTIM_ICFLT_CLOCK_DIV1
  – LL_LPTIM_ICFLT_CLOCK_DIV2
  – LL_LPTIM_ICFLT_CLOCK_DIV4
  – LL_LPTIM_ICFLT_CLOCK_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_LPTIM_IC_SetFilter
-
- CCMR1 IC2F LL_LPTIM_IC_SetFilter
-

LL_LPTIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetFilter (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the filter of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_ICFLT_CLOCK_DIV1
  – LL_LPTIM_ICFLT_CLOCK_DIV2
  – LL_LPTIM_ICFLT_CLOCK_DIV4
  – LL_LPTIM_ICFLT_CLOCK_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_LPTIM_IC_GetFilter
-
- CCMR1 IC2F LL_LPTIM_IC_GetFilter
-

LL_LPTIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_LPTIM_IC_SetPrescaler (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t Prescaler)

Function description

Set the prescaler of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2
- **Prescaler:** This parameter can be one of the following values:
  – LL_LPTIM_ICPSC_DIV1
  – LL_LPTIM_ICPSC_DIV2
  – LL_LPTIM_ICPSC_DIV4
  – LL_LPTIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_LPTIM_IC_SetPrescaler
-
- CCMR1 IC2PSC LL_LPTIM_IC_SetPrescaler
-

LL_LPTIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetPrescaler (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the prescaler of IC channels.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_ICPSC_DIV1
  – LL_LPTIM_ICPSC_DIV2
  – LL_LPTIM_ICPSC_DIV4
  – LL_LPTIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_LPTIM_IC_GetPrescaler
-
- CCMR1 IC2PSC LL_LPTIM_IC_GetPrescaler
-

LL_LPTIM_CC_SetChannelMode

Function name

__STATIC_INLINE void LL_LPTIM_CC_SetChannelMode (LPTIM_TypeDef * LPTIMx, uint32_t Channel, uint32_t CCMode)

Function description

Set the Channel Mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2
- **CCMode:** This parameter can be one of the following values:
  – LL_LPTIM_CCMODE_OUTPUT_PWM
  – LL_LPTIM_CCMODE_INPUTCAPTURE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1SEL LL_LPTIM_CC_SetChannelMode
- CCMR1 CC2SEL LL_LPTIM_CC_SetChannelMode

LL_LPTIM_CC_GetChannelMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_CC_GetChannelMode (const LPTIM_TypeDef * LPTIMx, uint32_t Channel)

Function description

Get the Channel Mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_LPTIM_CHANNEL_CH1
  – LL_LPTIM_CHANNEL_CH2

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_CCMODE_OUTPUT_PWM
  – LL_LPTIM_CCMODE_INPUTCAPTURE

Reference Manual to LL API cross reference:

- CCMR1 CC1SEL LL_LPTIM_CC_GetChannelMode
- CCMR1 CC2SEL LL_LPTIM_CC_GetChannelMode

LL_LPTIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetCaptureCH1 (const LPTIM_TypeDef * LPTIMx)

Function description

Get captured value for input channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset where offset can be retrieved by calling LL_LPTIM_IC_GET_OFFSET

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_LPTIM_IC_GetCaptureCH1

LL_LPTIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IC_GetCaptureCH2 (const LPTIM_TypeDef * LPTIMx)

Function description

Get captured value for input channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset where offset can be retrieved by calling LL_LPTIM_IC_GET_OFFSET

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_LPTIM_IC_GetCaptureCH2

LL_LPTIM_EnableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_EnableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Enable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- The first trigger event will start the timer, any successive trigger event will reset the counter and the timer will restart.
- The timeout value corresponds to the compare value; if no trigger occurs within the expected time frame, the MCU is waked-up by the compare match event.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_EnableTimeout

LL_LPTIM_DisableTimeout

Function name

__STATIC_INLINE void LL_LPTIM_DisableTimeout (LPTIM_TypeDef * LPTIMx)

Function description

Disable the timeout function.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- A trigger event arriving when the timer is already started will be ignored.

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_DisableTimeout

LL_LPTIM_IsEnabledTimeout

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledTimeout (const LPTIM_TypeDef * LPTIMx)

Function description

Indicate whether the timeout function is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR TIMOUT LL_LPTIM_IsEnabledTimeout

LL_LPTIM_TrigSw

Function name

__STATIC_INLINE void LL_LPTIM_TrigSw (LPTIM_TypeDef * LPTIMx)

Function description

Start the LPTIM counter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_TrigSw

LL_LPTIM_ConfigTrigger

Function name

__STATIC_INLINE void LL_LPTIM_ConfigTrigger (LPTIM_TypeDef * LPTIMx, uint32_t Source, uint32_t Filter, uint32_t Polarity)

Function description

Configure the external trigger used as a trigger event for the LPTIM.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **Source:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2 (*)
  - LL_LPTIM_TRIG_SOURCE_LPDMA_CH0_TCF
  - LL_LPTIM_TRIG_SOURCE_LPDMA_CH1_TCF
  - LL_LPTIM_TRIG_SOURCE_LPDMA_CH2_TCF
  - LL_LPTIM_TRIG_SOURCE_GPDMA_CH0_TCF
  - LL_LPTIM_TRIG_SOURCE_GPDMA_CH4_TCF

  (*) Value not defined in all devices.
-
- **Filter:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8
- **Polarity:** This parameter can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- An internal clock source must be present when a digital filter is required for the trigger.

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_ConfigTrigger
- CFGR TRGFLT LL_LPTIM_ConfigTrigger
- CFGR TRIGEN LL_LPTIM_ConfigTrigger

LL_LPTIM_GetTriggerSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_SOURCE_GPIO
  - LL_LPTIM_TRIG_SOURCE_RTCALARMA
  - LL_LPTIM_TRIG_SOURCE_RTCALARMB
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP1
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP2
  - LL_LPTIM_TRIG_SOURCE_RTCTAMP3
  - LL_LPTIM_TRIG_SOURCE_COMP1
  - LL_LPTIM_TRIG_SOURCE_COMP2 (*)
  - LL_LPTIM_TRIG_SOURCE_LPDMA_CH0_TCF
  - LL_LPTIM_TRIG_SOURCE_LPDMA_CH1_TCF
  - LL_LPTIM_TRIG_SOURCE_LPDMA_CH2_TCF
  - LL_LPTIM_TRIG_SOURCE_GPDMA_CH0_TCF
  - LL_LPTIM_TRIG_SOURCE_GPDMA_CH4_TCF

Reference Manual to LL API cross reference:

- CFGR TRIGSEL LL_LPTIM_GetTriggerSource

LL_LPTIM_GetTriggerFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_FILTER_NONE
  - LL_LPTIM_TRIG_FILTER_2
  - LL_LPTIM_TRIG_FILTER_4
  - LL_LPTIM_TRIG_FILTER_8

Reference Manual to LL API cross reference:

- CFGR TRGFLT LL_LPTIM_GetTriggerFilter

LL_LPTIM_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetTriggerPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual external trigger polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_TRIG_POLARITY_RISING
  - LL_LPTIM_TRIG_POLARITY_FALLING
  - LL_LPTIM_TRIG_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR TRIGEN LL_LPTIM_GetTriggerPolarity

LL_LPTIM_SetClockSource

Function name

__STATIC_INLINE void LL_LPTIM_SetClockSource (LPTIM_TypeDef * LPTIMx, uint32_t ClockSource)

Function description

Set the source of the clock used by the LPTIM instance.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_SetClockSource

LL_LPTIM_GetClockSource

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockSource (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual LPTIM instance clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPTIM_CLK_SOURCE_INTERNAL
  - LL_LPTIM_CLK_SOURCE_EXTERNAL

Reference Manual to LL API cross reference:

- CFGR CKSEL LL_LPTIM_GetClockSource

LL_LPTIM_ConfigClock

Function name

__STATIC_INLINE void LL_LPTIM_ConfigClock (LPTIM_TypeDef * LPTIMx, uint32_t ClockFilter, uint32_t ClockPolarity)

Function description

Configure the active edge or edges used by the counter when the LPTIM is clocked by an external clock source.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **ClockFilter:** This parameter can be one of the following values:
  – LL_LPTIM_CLK_FILTER_NONE
  – LL_LPTIM_CLK_FILTER_2
  – LL_LPTIM_CLK_FILTER_4
  – LL_LPTIM_CLK_FILTER_8
- **ClockPolarity:** This parameter can be one of the following values:
  – LL_LPTIM_CLK_POLARITY_RISING
  – LL_LPTIM_CLK_POLARITY_FALLING
  – LL_LPTIM_CLK_POLARITY_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- When both external clock signal edges are considered active ones, the LPTIM must also be clocked by an internal clock source with a frequency equal to at least four times the external clock frequency.
- An internal clock source must be present when a digital filter is required for external clock.

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_ConfigClock
- CFGR CKPOL LL_LPTIM_ConfigClock

LL_LPTIM_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockPolarity (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock polarity.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_CLK_POLARITY_RISING
  – LL_LPTIM_CLK_POLARITY_FALLING
  – LL_LPTIM_CLK_POLARITY_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetClockPolarity

LL_LPTIM_GetClockFilter

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetClockFilter (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual clock digital filter.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_CLK_FILTER_NONE
  – LL_LPTIM_CLK_FILTER_2
  – LL_LPTIM_CLK_FILTER_4
  – LL_LPTIM_CLK_FILTER_8

Reference Manual to LL API cross reference:

- CFGR CKFLT LL_LPTIM_GetClockFilter

LL_LPTIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_SetEncoderMode (LPTIM_TypeDef * LPTIMx, uint32_t EncoderMode)

Function description

Configure the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  – LL_LPTIM_ENCODER_MODE_RISING
  – LL_LPTIM_ENCODER_MODE_FALLING
  – LL_LPTIM_ENCODER_MODE_RISING_FALLING

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_SetEncoderMode

LL_LPTIM_GetEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_GetEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Get actual encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPTIM_ENCODER_MODE_RISING
  – LL_LPTIM_ENCODER_MODE_FALLING
  – LL_LPTIM_ENCODER_MODE_RISING_FALLING

Reference Manual to LL API cross reference:

- CFGR CKPOL LL_LPTIM_GetEncoderMode

LL_LPTIM_EnableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_EnableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Enable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.
- In this mode the LPTIM instance must be clocked by an internal clock source. Also, the prescaler division ratio must be equal to 1.
- LPTIM instance must be configured in continuous mode prior enabling the encoder mode.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_EnableEncoderMode

LL_LPTIM_DisableEncoderMode

Function name

__STATIC_INLINE void LL_LPTIM_DisableEncoderMode (LPTIM_TypeDef * LPTIMx)

Function description

Disable the encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Notes

- This function must be called when the LPTIM instance is disabled.

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_DisableEncoderMode

LL_LPTIM_IsEnabledEncoderMode

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledEncoderMode (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the LPTIM operates in encoder mode.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFGR ENC LL_LPTIM_IsEnabledEncoderMode

LL_LPTIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag for channel 1 (CC1CF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC1CF LL_LPTIM_ClearFlag_CC1

LL_LPTIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a capture/compare interrupt has occurred for channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC1IF LL_LPTIM_IsActiveFlag_CC1

LL_LPTIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare match flag for channel 2 (CC2CF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC2CF LL_LPTIM_ClearFlag_CC2

LL_LPTIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a capture/compare interrupt has occurred for channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC2IF LL_LPTIM_IsActiveFlag_CC2

LL_LPTIM_ClearFlag_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Clear the Capture/Compare 1 over-capture flag for channel 1 (CC1OCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC1OCF LL_LPTIM_ClearFlag_CC1O

LL_LPTIM_IsActiveFlag_CC1O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC1O (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a Capture/Compare 1 over-capture has occurred for channel 1.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC1OF LL_LPTIM_IsActiveFlag_CC1O

LL_LPTIM_ClearFlag_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Clear the Capture/Compare 2 over-capture flag for channel 2 (CC2OCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CC2OCF LL_LPTIM_ClearFlag_CC2O

LL_LPTIM_IsActiveFlag_CC2O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CC2O (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a Capture/Compare 2 over-capture has occurred for channel 2.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CC2OF LL_LPTIM_IsActiveFlag_CC2O

LL_LPTIM_ClearFlag_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload match flag (ARRMCF)

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARRMCF LL_LPTIM_ClearFlag_ARRM

LL_LPTIM_IsActiveFlag_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a autoreload match interrupt has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARRM LL_LPTIM_IsActiveFlag_ARRM

LL_LPTIM_ClearFlag_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Clear the external trigger valid edge flag(EXTTRIGCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR EXTTRIGCF LL_LPTIM_ClearFlag_EXTTRIG

LL_LPTIM_IsActiveFlag_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Inform application whether a valid edge on the selected external trigger input has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EXTTRIG LL_LPTIM_IsActiveFlag_EXTTRIG

LL_LPTIM_ClearFlag_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMP1OKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMP1OKCF LL_LPTIM_ClearFlag_CMP1OK

LL_LPTIM_IsActiveFlag_CMP1OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMP1OK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CCR1 register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMP1OK LL_LPTIM_IsActiveFlag_CMP1OK

LL_LPTIM_ClearFlag_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the compare register update interrupt flag (CMP2OKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMP2OKCF LL_LPTIM_ClearFlag_CMP2OK

LL_LPTIM_IsActiveFlag_CMP2OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_CMP2OK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_CCR2 register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMP2OK LL_LPTIM_IsActiveFlag_CMP2OK

LL_LPTIM_ClearFlag_DIEROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DIEROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the interrupt register update interrupt flag (DIEROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DIEROKCF LL_LPTIM_ClearFlag_DIEROK

LL_LPTIM_IsActiveFlag_DIEROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DIEROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_DIER register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DIEROK LL_LPTIM_IsActiveFlag_DIEROK

LL_LPTIM_ClearFlag_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the autoreload register update interrupt flag (ARROKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ARROKCF LL_LPTIM_ClearFlag_ARROK

LL_LPTIM_IsActiveFlag_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_ARR register has been successfully completed.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ARROK LL_LPTIM_IsActiveFlag_ARROK

LL_LPTIM_ClearFlag_UP

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UP (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to up interrupt flag (UPCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UPCF LL_LPTIM_ClearFlag_UP

LL_LPTIM_IsActiveFlag_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from down to up (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UP LL_LPTIM_IsActiveFlag_UP

LL_LPTIM_ClearFlag_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Clear the counter direction change to down interrupt flag (DOWNCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR DOWNCF LL_LPTIM_ClearFlag_DOWN

LL_LPTIM_IsActiveFlag_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Informs the application whether the counter direction has changed from up to down (when the LPTIM instance operates in encoder mode).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR DOWN LL_LPTIM_IsActiveFlag_DOWN

LL_LPTIM_ClearFlag_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Clear the repetition register update interrupt flag (REPOKCF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR REPOKCF LL_LPTIM_ClearFlag_REPOK

LL_LPTIM_IsActiveFlag_REPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_REPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the APB bus write operation to the LPTIMx_RCR register has been successfully completed; If so, a new one can be initiated.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REPOK LL_LPTIM_IsActiveFlag_REPOK

LL_LPTIM_ClearFlag_UE

Function name

__STATIC_INLINE void LL_LPTIM_ClearFlag_UE (LPTIM_TypeDef * LPTIMx)

Function description

Clear the update event flag (UECF).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR UECF LL_LPTIM_ClearFlag_UE

LL_LPTIM_IsActiveFlag_UE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsActiveFlag_UE (const LPTIM_TypeDef * LPTIMx)

Function description

Informs application whether the LPTIMx update event has occurred.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR UE LL_LPTIM_IsActiveFlag_UE

LL_LPTIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_EnableIT_CC1

LL_LPTIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_DisableIT_CC1

LL_LPTIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_LPTIM_IsEnabledIT_CC1

LL_LPTIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 interrupt (CC2IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_EnableIT_CC2

LL_LPTIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_DisableIT_CC2

LL_LPTIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_LPTIM_IsEnabledIT_CC2

LL_LPTIM_EnableIT_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 over-capture interrupt (CC1OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_EnableIT_CC1O

LL_LPTIM_DisableIT_CC1O

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC1O (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 over-capture interrupt (CC1OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_DisableIT_CC1O

LL_LPTIM_IsEnabledIT_CC1O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC1O (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 over-capture interrupt (CC1OIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1OIE LL_LPTIM_IsEnabledIT_CC1O

LL_LPTIM_EnableIT_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 over-capture interrupt (CC2OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_EnableIT_CC2O

LL_LPTIM_DisableIT_CC2O

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CC2O (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 over-capture interrupt (CC2OIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_DisableIT_CC2O

LL_LPTIM_IsEnabledIT_CC2O

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CC2O (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 over-capture interrupt (CC2OIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2OIE LL_LPTIM_IsEnabledIT_CC2O

LL_LPTIM_EnableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_EnableIT_ARRM

LL_LPTIM_DisableIT_ARRM

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARRM (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload match interrupt (ARRMIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_DisableIT_ARRM

LL_LPTIM_IsEnabledIT_ARRM

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARRM (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload match interrupt (ARRMIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER ARRMIE LL_LPTIM_IsEnabledIT_ARRM

LL_LPTIM_EnableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Enable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_EnableIT_EXTTRIG

LL_LPTIM_DisableIT_EXTTRIG

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_EXTTRIG (LPTIM_TypeDef * LPTIMx)

Function description

Disable external trigger valid edge interrupt (EXTTRIGIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_DisableIT_EXTTRIG

LL_LPTIM_IsEnabledIT_EXTTRIG

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_EXTTRIG (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates external trigger valid edge interrupt (EXTTRIGIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER EXTTRIGIE LL_LPTIM_IsEnabledIT_EXTTRIG

LL_LPTIM_EnableIT_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMP1OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP1OKIE LL_LPTIM_EnableIT_CMP1OK

LL_LPTIM_DisableIT_CMP1OK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMP1OK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMP1OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMPO1KIE LL_LPTIM_DisableIT_CMP1OK

LL_LPTIM_IsEnabledIT_CMP1OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMP1OK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMP1OKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMP1OKIE LL_LPTIM_IsEnabledIT_CMP1OK

LL_LPTIM_EnableIT_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Enable compare register write completed interrupt (CMP2OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_EnableIT_CMP2OK

LL_LPTIM_DisableIT_CMP2OK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_CMP2OK (LPTIM_TypeDef * LPTIMx)

Function description

Disable compare register write completed interrupt (CMP2OKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_DisableIT_CMP2OK

LL_LPTIM_IsEnabledIT_CMP2OK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_CMP2OK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the compare register write completed interrupt (CMP2OKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CMP2OKIE LL_LPTIM_IsEnabledIT_CMP2OK

LL_LPTIM_EnableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Enable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_EnableIT_ARROK

LL_LPTIM_DisableIT_ARROK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_ARROK (LPTIM_TypeDef * LPTIMx)

Function description

Disable autoreload register write completed interrupt (ARROKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_DisableIT_ARROK

LL_LPTIM_IsEnabledIT_ARROK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_ARROK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the autoreload register write completed interrupt (ARROKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER ARROKIE LL_LPTIM_IsEnabledIT_ARROK

LL_LPTIM_EnableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_EnableIT_UP

LL_LPTIM_DisableIT_UP

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UP (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to up interrupt (UPIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_DisableIT_UP

LL_LPTIM_IsEnabledIT_UP

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UP (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to up interrupt (UPIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER UPIE LL_LPTIM_IsEnabledIT_UP

LL_LPTIM_EnableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Enable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_EnableIT_DOWN

LL_LPTIM_DisableIT_DOWN

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_DOWN (LPTIM_TypeDef * LPTIMx)

Function description

Disable direction change to down interrupt (DOWNIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_DisableIT_DOWN

LL_LPTIM_IsEnabledIT_DOWN

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_DOWN (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the direction change to down interrupt (DOWNIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER DOWNIE LL_LPTIM_IsEnabledIT_DOWN

LL_LPTIM_EnableIT_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Enable repetition register update successfully completed interrupt (REPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_EnableIT_REPOK

LL_LPTIM_DisableIT_REPOK

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_REPOK (LPTIM_TypeDef * LPTIMx)

Function description

Disable repetition register update successfully completed interrupt (REPOKIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_DisableIT_REPOK

LL_LPTIM_IsEnabledIT_REPOK

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_REPOK (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the repetition register update successfully completed interrupt (REPOKIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER REPOKIE LL_LPTIM_IsEnabledIT_REPOK

LL_LPTIM_EnableIT_UE

Function name

__STATIC_INLINE void LL_LPTIM_EnableIT_UE (LPTIM_TypeDef * LPTIMx)

Function description

Enable update event interrupt (UEIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_EnableIT_UE

LL_LPTIM_DisableIT_UE

Function name

__STATIC_INLINE void LL_LPTIM_DisableIT_UE (LPTIM_TypeDef * LPTIMx)

Function description

Disable update event interrupt (UEIE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_DisableIT_UE

LL_LPTIM_IsEnabledIT_UE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledIT_UE (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the update event interrupt (UEIE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance @ retval State of bit(1 or 0).

Reference Manual to LL API cross reference:

- DIER UEIE LL_LPTIM_IsEnabledIT_UE

LL_LPTIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_UPDATE (LPTIM_TypeDef * LPTIMx)

Function description

Enable update DMA request.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_EnableDMAReq_UPDATE

LL_LPTIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_UPDATE (LPTIM_TypeDef * LPTIMx)

Function description

Disable update DMA request.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_DisableDMAReq_UPDATE

LL_LPTIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_UPDATE (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the update DMA request is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UEDE LL_LPTIM_IsEnabledDMAReq_UPDATE

LL_LPTIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_EnableDMAReq_CC1

LL_LPTIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_CC1 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_DisableDMAReq_CC1

LL_LPTIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_CC1 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_LPTIM_IsEnabledDMAReq_CC1

LL_LPTIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_LPTIM_EnableDMAReq_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_EnableDMAReq_CC2

LL_LPTIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_LPTIM_DisableDMAReq_CC2 (LPTIM_TypeDef * LPTIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_DisableDMAReq_CC2

LL_LPTIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_LPTIM_IsEnabledDMAReq_CC2 (const LPTIM_TypeDef * LPTIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **LPTIMx:** Low-Power Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_LPTIM_IsEnabledDMAReq_CC2

109.3 LPTIM Firmware driver defines

The following section lists the various define and macros of the module.

109.3.1 LPTIM

LPTIM

Input1 Source

LL_LPTIM_INPUT1_SRC_GPIO

For LPTIM1, LPTIM2, LPTIM3 and LPTIM4

LL_LPTIM_INPUT1_SRC_COMP1

For LPTIM1, LPTIM2, LPTIM3 and LPTIM4

LL_LPTIM_INPUT1_SRC_COMP2

For LPTIM1 and LPTIM2

Input2 Source

LL_LPTIM_INPUT2_SRC_GPIO
For LPTIM1 and LPTIM2

LL_LPTIM_INPUT2_SRC_COMP2
For LPTIM1 and LPTIM2

LPTIM1 Input Ch1 Remap

LL_LPTIM_LPTIM1_IC1_RMP_GPIO
IC1 connected to GPIO

LL_LPTIM_LPTIM1_IC1_RMP_COMP1
IC1 connected to COMP1

LL_LPTIM_LPTIM1_IC1_RMP_COMP2
IC1 connected to COMP2

LPTIM1 Input Ch2 Remap

LL_LPTIM_LPTIM1_IC2_RMP_GPIO
IC2 connected to GPIO

LL_LPTIM_LPTIM1_IC2_RMP_LSI
IC2 connected to LSI

LL_LPTIM_LPTIM1_IC2_RMP_LSE
IC2 connected to LSE

LPTIM2 Input Ch1 Remap

LL_LPTIM_LPTIM2_IC1_RMP_GPIO
IC1 connected to GPIO

LL_LPTIM_LPTIM2_IC1_RMP_COMP1
IC1 connected to COMP1

LL_LPTIM_LPTIM2_IC1_RMP_COMP2
IC1 connected to COMP2

LPTIM2 Input Ch2 Remap

LL_LPTIM_LPTIM2_IC2_RMP_GPIO
IC2 connected to GPIO

LL_LPTIM_LPTIM2_IC2_RMP_HSI
IC2 connected to HSI

LL_LPTIM_LPTIM2_IC2_RMP_MSIS_1024
IC2 connected to MSIS/1024

LL_LPTIM_LPTIM2_IC2_RMP_MSIS_4
IC2 connected to MSIS/4

LPTIM3 Input Ch1 Remap

LL_LPTIM_LPTIM3_IC1_RMP_GPIO
IC1 connected to GPIO

LL_LPTIM_LPTIM3_IC1_RMP_COMP1

IC1 connected to COMP1

LL_LPTIM_LPTIM3_IC1_RMP_COMP2

IC1 connected to COMP2

Clock Filter

LL_LPTIM_CLK_FILTER_NONE

Any external clock signal level change is considered as a valid transition

LL_LPTIM_CLK_FILTER_2

External clock signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_4

External clock signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_CLK_FILTER_8

External clock signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Clock Polarity

LL_LPTIM_CLK_POLARITY_RISING

The rising edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_CLK_POLARITY_RISING_FALLING

Both edges are active edges

Clock Source

LL_LPTIM_CLK_SOURCE_INTERNAL

LPTIM is clocked by internal clock source (APB clock or any of the embedded oscillators)

LL_LPTIM_CLK_SOURCE_EXTERNAL

LPTIM is clocked by an external clock source through the LPTIM external Input1

Counter Mode

LL_LPTIM_COUNTER_MODE_INTERNAL

The counter is incremented following each internal clock pulse

LL_LPTIM_COUNTER_MODE_EXTERNAL

The counter is incremented following each valid clock pulse on the LPTIM external Input1

Encoder Mode

LL_LPTIM_ENCODER_MODE_RISING

The rising edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_FALLING

The falling edge is the active edge used for counting

LL_LPTIM_ENCODER_MODE_RISING_FALLING

Both edges are active edges

Get Flags Defines

LL_LPTIM_ISR_CMP1OK

Compare register 1 update OK

LL_LPTIM_ISR_CMP2OK

Compare register 2 update OK

LL_LPTIM_ISR_CC1IF

Capture/Compare 1 interrupt flag

LL_LPTIM_ISR_CC2IF

Capture/Compare 2 interrupt flag

LL_LPTIM_ISR_CC1OF

Capture/Compare 1 over-capture flag

LL_LPTIM_ISR_CC2OF

Capture/Compare 2 over-capture flag

LL_LPTIM_ISR_DIEROK

Interrupt enable register update OK

LL_LPTIM_ISR_ARRM

Autoreload match

LL_LPTIM_ISR_EXTTRIG

External trigger edge event

LL_LPTIM_ISR_ARROK

Autoreload register update OK

LL_LPTIM_ISR_UP

Counter direction change down to up

LL_LPTIM_ISR_DOWN

Counter direction change up to down

LL_LPTIM_ISR_UE

Update event

LL_LPTIM_ISR_REPOK

Repetition register update OK

IT Defines

LL_LPTIM_DIER_CMP1OKIE

Compare register 1 update OK

LL_LPTIM_DIER_CMP2OKIE

Compare register 2 update OK

LL_LPTIM_DIER_CC1IFIE

Capture/Compare 1 interrupt flag

LL_LPTIM_DIER_CC2IFIE

Capture/Compare 2 interrupt flag

LL_LPTIM_DIER_CC1OFIE

Capture/Compare 1 over-capture flag

LL_LPTIM_DIER_CC2OFIE

Capture/Compare 2 over-capture flag

LL_LPTIM_DIER_ARRMIE

Autoreload match

LL_LPTIM_DIER_EXTTRIGIE

External trigger edge event

LL_LPTIM_DIER_ARROKIE

Autoreload register update OK

LL_LPTIM_DIER_UPIE

Counter direction change down to up

LL_LPTIM_DIER_DOWNIE

Counter direction change up to down

LL_LPTIM_DIER_UEIE

Update event

LL_LPTIM_DIER_REPOKIE

Repetition register update OK

Input Capture Filter

LL_LPTIM_ICFLT_CLOCK_DIV1

any external input capture signal level change is considered as a valid transition

LL_LPTIM_ICFLT_CLOCK_DIV2

external input capture signal level change must be stable for at least 2 clock periods before it is considered as valid transition

LL_LPTIM_ICFLT_CLOCK_DIV4

external input capture signal level change must be stable for at least 4 clock periods before it is considered as valid transition

LL_LPTIM_ICFLT_CLOCK_DIV8

external input capture signal level change must be stable for at least 8 clock periods before it is considered as valid transition

Input Capture Polarity

LL_LPTIM_ICPOLARITY_RISING

Capture/Compare input rising polarity

LL_LPTIM_ICPOLARITY_FALLING

Capture/Compare input falling polarity

LL_LPTIM_ICPOLARITY_RISING_FALLING

Capture/Compare input rising and falling polarities

Input Capture Prescaler

LL_LPTIM_ICPSC_DIV1

Capture performed each time an edge is detected on the capture input

LL_LPTIM_ICPSC_DIV2

Capture performed once every 2 events

LL_LPTIM_ICPSC_DIV4

Capture performed once every 4 events

LL_LPTIM_ICPSC_DIV8

Capture performed once every 8 events

Input Capture selection

LL_LPTIM_CCMODE_OUTPUT_PWM

Select PWM mode

LL_LPTIM_CCMODE_INPUTCAPTURE

Select Input Capture mode

Operating Mode

LL_LPTIM_OPERATING_MODE_CONTINUOUS

LP Timer starts in continuous mode

LL_LPTIM_OPERATING_MODE_ONESHOT

LP Tilmer starts in single mode

Output Polarity

LL_LPTIM_OUTPUT_POLARITY_REGULAR

The LPTIM output reflects the compare results between LPTIMx_ARR and LPTIMx_CCRx registers

LL_LPTIM_OUTPUT_POLARITY_INVERSE

The LPTIM output reflects the inverse of the compare results between LPTIMx_ARR and LPTIMx_CCx registers

Output Waveform Type

LL_LPTIM_OUTPUT_WAVEFORM_PWM

LPTIM generates either a PWM waveform or a One pulse waveform depending on chosen operating mode CONTINUOUS or SINGLE

LL_LPTIM_OUTPUT_WAVEFORM_SETONCE

LPTIM generates a Set Once waveform

Prescaler Value

LL_LPTIM_PRESCALER_DIV1

Prescaler division factor is set to 1

LL_LPTIM_PRESCALER_DIV2

Prescaler division factor is set to 2

LL_LPTIM_PRESCALER_DIV4

Prescaler division factor is set to 4

LL_LPTIM_PRESCALER_DIV8

Prescaler division factor is set to 8

LL_LPTIM_PRESCALER_DIV16

Prescaler division factor is set to 16

LL_LPTIM_PRESCALER_DIV32

Prescaler division factor is set to 32

LL_LPTIM_PRESCALER_DIV64

Prescaler division factor is set to 64

LL_LPTIM_PRESCALER_DIV128

Prescaler division factor is set to 128

Trigger Filter

LL_LPTIM_TRIG_FILTER_NONE

Any trigger active level change is considered as a valid trigger

LL_LPTIM_TRIG_FILTER_2

Trigger active level change must be stable for at least 2 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_4

Trigger active level change must be stable for at least 4 clock periods before it is considered as valid trigger

LL_LPTIM_TRIG_FILTER_8

Trigger active level change must be stable for at least 8 clock periods before it is considered as valid trigger

Trigger Polarity

LL_LPTIM_TRIG_POLARITY_RISING

LPTIM counter starts when a rising edge is detected

LL_LPTIM_TRIG_POLARITY_FALLING

LPTIM counter starts when a falling edge is detected

LL_LPTIM_TRIG_POLARITY_RISING_FALLING

LPTIM counter starts when a rising or a falling edge is detected

Trigger Source

LL_LPTIM_TRIG_SOURCE_GPIO

External input trigger is connected to TIMx_ETR input

LL_LPTIM_TRIG_SOURCE_RTCALARMA

External input trigger is connected to RTC Alarm A

LL_LPTIM_TRIG_SOURCE_RTCALARMB

External input trigger is connected to RTC Alarm B

LL_LPTIM_TRIG_SOURCE_RTCTAMP1

External input trigger is connected to RTC Tamper 1

LL_LPTIM_TRIG_SOURCE_RTCTAMP2

External input trigger is connected to RTC Tamper 2

LL_LPTIM_TRIG_SOURCE_RTCTAMP3

External input trigger is connected to RTC Tamper 3

LL_LPTIM_TRIG_SOURCE_COMP1

External input trigger is connected to COMP1 output

LL_LPTIM_TRIG_SOURCE_COMP2

External input trigger is connected to COMP2 output

LL_LPTIM_TRIG_SOURCE_LPDMA_CH0_TCF

External input trigger is connected to LPDMA CH0 transfer complete

LL_LPTIM_TRIG_SOURCE_LPDMA_CH1_TCF

External input trigger is connected to LPDMA CH1 transfer complete

LL_LPTIM_TRIG_SOURCE_LPDMA_CH2_TCF

External input trigger is connected to LPDMA CH2 transfer complete

LL_LPTIM_TRIG_SOURCE_GPDMA_CH0_TCF

External input trigger is connected to GPDMA CH0 transfer complete

LL_LPTIM_TRIG_SOURCE_GPDMA_CH4_TCF

External input trigger is connected to GPDMA CH4 transfer complete

Update Mode

LL_LPTIM_UPDATE_MODE_IMMEDIATE

Preload is disabled: registers are updated after each APB bus write access

LL_LPTIM_UPDATE_MODE_ENDOFPERIOD

preload is enabled: registers are updated at the end of the current LPTIM period

Common Write and read registers Macros

LL_LPTIM_WriteReg

Description:

- Write a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPTIM_ReadReg

Description:

- Read a value in LPTIM register.

Parameters:

- __INSTANCE__: LPTIM Instance
- __REG__: Register to be read

Return value:

- Register: value

LL_LPTIM_IC_GET_OFFSET

Description:

- LPTimer Input Capture Get Offset(in counter step unit)

Parameters:

- __PSC__: This parameter can be one of the following values:
  - LL_LPTIM_PRESCALER_DIV1
  - LL_LPTIM_PRESCALER_DIV2
  - LL_LPTIM_PRESCALER_DIV4
  - LL_LPTIM_PRESCALER_DIV8
  - LL_LPTIM_PRESCALER_DIV16
  - LL_LPTIM_PRESCALER_DIV32
  - LL_LPTIM_PRESCALER_DIV64
  - LL_LPTIM_PRESCALER_DIV128
- __FLT__: This parameter can be one of the following values:
  - LL_LPTIM_ICFLT_CLOCK_DIV1
  - LL_LPTIM_ICFLT_CLOCK_DIV2
  - LL_LPTIM_ICFLT_CLOCK_DIV4
  - LL_LPTIM_ICFLT_CLOCK_DIV8

Return value:

- offset: value

Notes:

- The real capture value corresponding to the input capture trigger can be calculated using the formula hereafter : Real capture value = captured(LPTIM_CCRx) - offset The Offset value is depending on the glitch filter value for the channel and the value of the prescaler for the kernel clock. Please check Errata Sheet V1_8 for more details under "variable latency on input capture channel" section.

110 LL LPUART Generic Driver

110.1 LPUART Firmware driver registers structures

110.1.1 LL_LPUART_InitTypeDef

LL_LPUART_InitTypeDef is defined in the stm32u5xx_ll_lpuart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***LPUART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***
  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***LPUART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***LPUART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***LPUART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***LPUART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

110.2 LPUART Firmware driver API description

The following section lists the various functions of the LPUART library.

110.2.1 Detailed description of functions

LL_LPUART_Enable

Function name

__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)

Function description

LPUART Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name

__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)

Function description

LPUART Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.
- In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name

__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name

__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (const USART_TypeDef * LPUARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  – LL_LPUART_FIFOTHRESHOLD_1_8
  – LL_LPUART_FIFOTHRESHOLD_1_4
  – LL_LPUART_FIFOTHRESHOLD_1_2
  – LL_LPUART_FIFOTHRESHOLD_3_4
  – LL_LPUART_FIFOTHRESHOLD_7_8
  – LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_FIFOTHRESHOLD_1_8
  – LL_LPUART_FIFOTHRESHOLD_1_4
  – LL_LPUART_FIFOTHRESHOLD_1_2
  – LL_LPUART_FIFOTHRESHOLD_3_4
  – LL_LPUART_FIFOTHRESHOLD_7_8
  – LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **TXThreshold:** This parameter can be one of the following values:
  – LL_LPUART_FIFOTHRESHOLD_1_8
  – LL_LPUART_FIFOTHRESHOLD_1_4
  – LL_LPUART_FIFOTHRESHOLD_1_2
  – LL_LPUART_FIFOTHRESHOLD_3_4
  – LL_LPUART_FIFOTHRESHOLD_7_8
  – LL_LPUART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  – LL_LPUART_FIFOTHRESHOLD_1_8
  – LL_LPUART_FIFOTHRESHOLD_1_4
  – LL_LPUART_FIFOTHRESHOLD_1_2
  – LL_LPUART_FIFOTHRESHOLD_3_4
  – LL_LPUART_FIFOTHRESHOLD_7_8
  – LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART enabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART disabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is disabled, LPUART is not able to wake up the MCU from Stop mode

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name

__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_SetTransferDirection
- CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (const USART_TypeDef * LPUARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_DIRECTION_NONE
  – LL_LPUART_DIRECTION_RX
  – LL_LPUART_DIRECTION_TX
  – LL_LPUART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_GetTransferDirection
- CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name

__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance
- **Parity:** This parameter can be one of the following values:
  – LL_LPUART_PARITY_NONE
  – LL_LPUART_PARITY_EVEN
  – LL_LPUART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_SetParity
- CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetParity (const USART_TypeDef * LPUARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_GetParity
- CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance
- **Method:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (const USART_TypeDef * LPUARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name

__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)

Function description

Set Word length (nb of data bits, excluding start and stop bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (const USART_TypeDef * LPUARTx)

Function description

Return Word length (i.e.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)

Function description

Prevent Mute Mode use.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name

__STATIC_INLINE void LL_LPUART_SetPrescaler (USART_TypeDef * LPUARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  – LL_LPUART_PRESCALER_DIV1
  – LL_LPUART_PRESCALER_DIV2
  – LL_LPUART_PRESCALER_DIV4
  – LL_LPUART_PRESCALER_DIV6
  – LL_LPUART_PRESCALER_DIV8
  – LL_LPUART_PRESCALER_DIV10
  – LL_LPUART_PRESCALER_DIV12
  – LL_LPUART_PRESCALER_DIV16
  – LL_LPUART_PRESCALER_DIV32
  – LL_LPUART_PRESCALER_DIV64
  – LL_LPUART_PRESCALER_DIV128
  – LL_LPUART_PRESCALER_DIV256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (const USART_TypeDef * LPUARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_PRESCALER_DIV1
  – LL_LPUART_PRESCALER_DIV2
  – LL_LPUART_PRESCALER_DIV4
  – LL_LPUART_PRESCALER_DIV6
  – LL_LPUART_PRESCALER_DIV8
  – LL_LPUART_PRESCALER_DIV10
  – LL_LPUART_PRESCALER_DIV12
  – LL_LPUART_PRESCALER_DIV16
  – LL_LPUART_PRESCALER_DIV32
  – LL_LPUART_PRESCALER_DIV64
  – LL_LPUART_PRESCALER_DIV128
  – LL_LPUART_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance
- **StopBits:** This parameter can be one of the following values:
  – LL_LPUART_STOPBITS_1
  – LL_LPUART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (const USART_TypeDef * LPUARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_STOPBITS_1
  – LL_LPUART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name

__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_ConfigCharacter
- CR1 PCE LL_LPUART_ConfigCharacter
- CR1 M LL_LPUART_ConfigCharacter
- CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **LPUARTx:** LPUART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_LPUART_TXRX_STANDARD
  - LL_LPUART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_TXRX_STANDARD
  – LL_LPUART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  – LL_LPUART_RXPIN_LEVEL_STANDARD
  – LL_LPUART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_RXPIN_LEVEL_STANDARD
  – LL_LPUART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  – LL_LPUART_TXPIN_LEVEL_STANDARD
  – LL_LPUART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_TXPIN_LEVEL_STANDARD
  – LL_LPUART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **LPUARTx:** LPUART Instance
- **DataLogic:** This parameter can be one of the following values:
  – LL_LPUART_BINARY_LOGIC_POSITIVE
  – LL_LPUART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (const USART_TypeDef * LPUARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_BINARY_LOGIC_POSITIVE
  – LL_LPUART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance
- **BitOrder:** This parameter can be one of the following values:
  – LL_LPUART_BITORDER_LSBFIRST
  – LL_LPUART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

LL_LPUART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (const USART_TypeDef * LPUARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_BITORDER_LSBFIRST
  – LL_LPUART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the LPUART node.

Parameters

- **LPUARTx:** LPUART Instance
- **AddressLen:** This parameter can be one of the following values:
  – LL_LPUART_ADDRESS_DETECT_4B
  – LL_LPUART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the LPUART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_ConfigNodeAddress
- CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (const USART_TypeDef * LPUARTx)

Function description

Return 8 bit Address of the LPUART node as set in ADD field of CR2.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (const USART_TypeDef * LPUARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  – LL_LPUART_HWCONTROL_NONE
  – LL_LPUART_HWCONTROL_RTS
  – LL_LPUART_HWCONTROL_CTS
  – LL_LPUART_HWCONTROL_RTS_CTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_SetHWFlowCtrl
- CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (const USART_TypeDef * LPUARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_HWCONTROL_NONE
  – LL_LPUART_HWCONTROL_RTS
  – LL_LPUART_HWCONTROL_CTS
  – LL_LPUART_HWCONTROL_RTS_CTS

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_GetHWFlowCtrl
- CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (const USART_TypeDef * LPUARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL_LPUART_SetBaudRate

Function name

__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)

Function description

Configure LPUART BRR register for achieving expected Baud Rate value.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
- Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value ! = 0).
- Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (const USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)

Function description

Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  – LL_LPUART_PRESCALER_DIV1
  – LL_LPUART_PRESCALER_DIV2
  – LL_LPUART_PRESCALER_DIV4
  – LL_LPUART_PRESCALER_DIV6
  – LL_LPUART_PRESCALER_DIV8
  – LL_LPUART_PRESCALER_DIV10
  – LL_LPUART_PRESCALER_DIV12
  – LL_LPUART_PRESCALER_DIV16
  – LL_LPUART_PRESCALER_DIV32
  – LL_LPUART_PRESCALER_DIV64
  – LL_LPUART_PRESCALER_DIV128
  – LL_LPUART_PRESCALER_DIV256

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (const USART_TypeDef * LPUARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : c

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_SetDEAssertionTime

LL_LPUART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name

__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name

__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_IsEnabledDEMode

LL_LPUART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  – LL_LPUART_DE_POLARITY_HIGH
  – LL_LPUART_DE_POLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (const USART_TypeDef * LPUARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_DE_POLARITY_HIGH
  – LL_LPUART_DE_POLARITY_LOW

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Framing Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Noise error detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART OverRun Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE line detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS interrupt Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Busy Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Send Break Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)

Function description

Clear Parity Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)

Function description

Clear Framing Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)

Function description

Clear Noise detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)

Function description

Clear Character Match Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Enable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Enable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Disable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Disable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE Interrupt source is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * LPUARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (const USART_TypeDef * LPUARTx, uint32_t Direction)

Function description

Get the LPUART data register address used for DMA transfer.

Parameters

- **LPUARTx:** LPUART Instance
- **Direction:** This parameter can be one of the following values:
  – LL_LPUART_DMA_REG_DATA_TRANSMIT
  – LL_LPUART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_DMA_GetRegAddr
-
- TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name

__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name

__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name

__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)

Function description

Request Break sending.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)

Function description

Put LPUART in mute mode and set the RWU flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestTxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Transmit data FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- TXFRQ bit is set to flush the whole FIFO when FIFO mode is enabled. This also sets the flag TXFE (TXFIFO empty bit in the LPUART_ISR register).
- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_LPUART_RequestTxDataFlush

LL_LPUART_Enable_SelectedTrigger

Function name

__STATIC_INLINE void LL_LPUART_Enable_SelectedTrigger (USART_TypeDef * LPUARTx)

Function description

Enable Selected Trigger.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_LPUART_Enable_SelectedTrigger

LL_LPUART_Disable_SelectedTrigger

Function name

__STATIC_INLINE void LL_LPUART_Disable_SelectedTrigger (USART_TypeDef * LPUARTx)

Function description

Disable Selected Trigger.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_LPUART_Disable_SelectedTrigger

LL_LPUART_IsEnabled_SelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled_SelectedTrigger (const USART_TypeDef * LPUARTx)

Function description

Indicate if selected Trigger is disabled or enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_LPUART_IsEnabled_SelectedTrigger

LL_LPUART_Enable_AutonomousSendIdleFrame

Function name

__STATIC_INLINE void LL_LPUART_Enable_AutonomousSendIdleFrame (USART_TypeDef * LPUARTx)

Function description

Enable Autonomous Send Idle Frame feature.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR IDLEDIS LL_LPUART_Enable_AutonomousSendIdleFrame

LL_LPUART_Disable_AutonomousSendIdleFrame

Function name

__STATIC_INLINE void LL_LPUART_Disable_AutonomousSendIdleFrame (USART_TypeDef * LPUARTx)

Function description

Disable Autonomous Send Idle Frame feature.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR IDLEDIS LL_LPUART_Disable_AutonomousSendIdleFrame

LL_LPUART_IsEnabled_AutonomousSendIdleFrame

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled_AutonomousSendIdleFrame (const USART_TypeDef * LPUARTx)

Function description

Indicate if Autonomous send Idle Frame feature is disabled or enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AUTOCR IDLEDIS LL_LPUART_IsEnabled_AutonomousSendIdleFrame

LL_LPUART_SetNbTxData

Function name

__STATIC_INLINE void LL_LPUART_SetNbTxData (USART_TypeDef * LPUARTx, uint32_t Nbdata)

Function description

Configure the Number of transferred data in bytes.

Parameters

- **LPUARTx:** LPUART Instance
- **Nbdata:** This parameter can be a value between 0 and 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TDN LL_LPUART_SetNbTxData

LL_LPUART_GetNbTxData

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNbTxData (const USART_TypeDef * LPUARTx)

Function description

Retrieve the Number of transferred data in bytes.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be a value between 0 and 0xFFFF

Reference Manual to LL API cross reference:

- AUTOCR TDN LL_LPUART_GetNbTxData

LL_LPUART_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetTriggerPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Set the trigger polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  – LL_LPUART_TRIG_POLARITY_RISING
  – LL_LPUART_TRIG_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_LPUART_SetTriggerPolarity

LL_LPUART_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTriggerPolarity (const USART_TypeDef * LPUARTx)

Function description

Get the trigger polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_TRIG_POLARITY_RISING
  – LL_LPUART_TRIG_POLARITY_FALLING

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_LPUART_GetTriggerPolarity

LL_LPUART_SetSelectedTrigger

Function name

__STATIC_INLINE void LL_LPUART_SetSelectedTrigger (USART_TypeDef * LPUARTx, uint32_t Trigger)

Function description

Set the selected trigger.

Parameters

- **LPUARTx:** LPUART Instance
- **Trigger:** This parameter can be one of the following values:
  – LL_LPUART_LPDMA1_CH0_TCF_TRG
  – LL_LPUART_LPDMA1_CH1_TCF_TRG
  – LL_LPUART_LPDMA1_CH2_TCF_TRG
  – LL_LPUART_LPDMA1_CH3_TCF_TRG
  – LL_LPUART_EXTI_LINE6_TRG
  – LL_LPUART_EXTI_LINE8_TRG
  – LL_LPUART_LPTIM1_OUT_TRG
  – LL_LPUART_LPTIM3_OUT_TRG
  – LL_LPUART_COMP1_OUT_TRG
  – LL_LPUART_COMP2_OUT_TRG
  – LL_LPUART_RTC_ALRA_TRG
  – LL_LPUART_RTC_WUT_TRG

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_LPUART_SetSelectedTrigger

LL_LPUART_GetSelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetSelectedTrigger (const USART_TypeDef * LPUARTx)

Function description

Get the selected trigger.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_LPDMA1_CH0_TCF_TRG
  – LL_LPUART_LPDMA1_CH1_TCF_TRG
  – LL_LPUART_LPDMA1_CH2_TCF_TRG
  – LL_LPUART_LPDMA1_CH3_TCF_TRG
  – LL_LPUART_EXTI_LINE6_TRG
  – LL_LPUART_EXTI_LINE8_TRG
  – LL_LPUART_LPTIM1_OUT_TRG
  – LL_LPUART_LPTIM3_OUT_TRG
  – LL_LPUART_COMP1_OUT_TRG
  – LL_LPUART_COMP2_OUT_TRG
  – LL_LPUART_RTC_ALRA_TRG
  – LL_LPUART_RTC_WUT_TRG

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_LPUART_GetSelectedTrigger

LL_LPUART_DeInit

Function name

ErrorStatus LL_LPUART_DeInit (const USART_TypeDef * LPUARTx)

Function description

De-initialize LPUART registers (Registers restored to their default values).

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are de-initialized
  - ERROR: not applicable

LL_LPUART_Init

Function name

ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, const LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.

Parameters

- **LPUARTx:** LPUART Instance
- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content
  - ERROR: Problem occurred during LPUART Registers initialization

Notes

- As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name

void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Set each LL_LPUART_InitTypeDef field to default value.

Parameters

- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

110.3 LPUART Firmware driver defines

The following section lists the various define and macros of the module.

110.3.1 LPUART

LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_LPUART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Autonomous Trigger Polarity

LL_LPUART_TRIG_POLARITY_RISING

LPUART triggered on rising edge

LL_LPUART_TRIG_POLARITY_FALLING

LPUART triggered on falling edge

Autonomous Trigger selection

LL_LPUART_LPDMA1_CH0_TCF_TRG

LPUART LPDMA1 channel0 Internal Trigger

LL_LPUART_LPDMA1_CH1_TCF_TRG

LPUART LPDMA1 channel1 Internal Trigger

LL_LPUART_LPDMA1_CH2_TCF_TRG

LPUART LPDMA1 channel2 Internal Trigger

LL_LPUART_LPDMA1_CH3_TCF_TRG

LPUART LPDMA1 channel3 Internal Trigger

LL_LPUART_EXTI_LINE6_TRG

LPUART EXTI line 6 Internal Trigger

LL_LPUART_EXTI_LINE8_TRG

LPUART EXTI line 8 Internal Trigger

LL_LPUART_LPTIM1_OUT_TRG

LPUART LPTIM1 out Internal Trigger

LL_LPUART_LPTIM3_OUT_TRG

LPUART LPTIM3 out Internal Trigger

LL_LPUART_COMP1_OUT_TRG

LPUART COMP1 out Internal Trigger

LL_LPUART_COMP2_OUT_TRG

LPUART COMP2 out Internal Trigger

LL_LPUART_RTC_ALRA_TRG

LPUART RTC alarm Internal Trigger

LL_LPUART_RTC_WUT_TRG

LPUART RTC wakeup Internal Trigger

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_LPUART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_LPUART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF

Parity error clear flag

LL_LPUART_ICR_FECF

Framing error clear flag

LL_LPUART_ICR_NCF

Noise error detected clear flag

LL_LPUART_ICR_ORECF

Overrun error clear flag

LL_LPUART_ICR_IDLECF

Idle line detected clear flag

LL_LPUART_ICR_TCCF

Transmission complete clear flag

LL_LPUART_ICR_CTSCF

CTS clear flag

LL_LPUART_ICR_CMCF

Character match clear flag

Datawidth

LL_LPUART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_LPUART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_LPUART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH

DE signal is active high

LL_LPUART_DE_POLARITY_LOW

DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_LPUART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_LPUART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_LPUART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_LPUART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_LPUART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_LPUART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_LPUART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_LPUART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_LPUART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE

Parity error flag

LL_LPUART_ISR_FE

Framing error flag

LL_LPUART_ISR_NE

Noise detected flag

LL_LPUART_ISR_ORE

Overrun error flag

LL_LPUART_ISR_IDLE

Idle line detected flag

LL_LPUART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_LPUART_ISR_TC

Transmission complete flag

LL_LPUART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_LPUART_ISR_CTSIF

CTS interrupt flag

LL_LPUART_ISR_CTS

CTS flag

LL_LPUART_ISR_BUSY

Busy flag

LL_LPUART_ISR_CMF

Character match flag

LL_LPUART_ISR_SBKF

Send break flag

LL_LPUART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_LPUART_ISR_TEACK

Transmit enable acknowledge flag

LL_LPUART_ISR_REACK

Receive enable acknowledge flag

LL_LPUART_ISR_TXFE

TX FIFO empty flag

LL_LPUART_ISR_RXFF

RX FIFO full flag

LL_LPUART_ISR_RXFT

RX FIFO threshold flag

LL_LPUART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_LPUART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_LPUART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_LPUART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE

IDLE interrupt enable

LL_LPUART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_LPUART_CR1_TCIE

Transmission complete interrupt enable

LL_LPUART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_LPUART_CR1_PEIE

Parity error

LL_LPUART_CR1_CMIE

Character match interrupt enable

LL_LPUART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_LPUART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_LPUART_CR3_EIE

Error interrupt enable

LL_LPUART_CR3_CTSIE

CTS interrupt enable

LL_LPUART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_LPUART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE

Parity control disabled

LL_LPUART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1

Input clock not divided

LL_LPUART_PRESCALER_DIV2

Input clock divided by 2

LL_LPUART_PRESCALER_DIV4

Input clock divided by 4

LL_LPUART_PRESCALER_DIV6

Input clock divided by 6

LL_LPUART_PRESCALER_DIV8

Input clock divided by 8

LL_LPUART_PRESCALER_DIV10

Input clock divided by 10

LL_LPUART_PRESCALER_DIV12

Input clock divided by 12

LL_LPUART_PRESCALER_DIV16

Input clock divided by 16

LL_LPUART_PRESCALER_DIV32

Input clock divided by 32

LL_LPUART_PRESCALER_DIV64

Input clock divided by 64

LL_LPUART_PRESCALER_DIV128

Input clock divided by 128

LL_LPUART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_LPUART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1

1 stop bit

LL_LPUART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_LPUART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_LPUART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE

LPUART wake up from Mute mode on Idle Line

LL_LPUART_WAKEUP_ADDRESSMARK

LPUART wake up from Mute mode on Address Mark

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

111 LL OPAMP Generic Driver

111.1 OPAMP Firmware driver registers structures

111.1.1 LL_OPAMP_InitTypeDef

LL_OPAMP_InitTypeDef is defined in the stm32u5xx_ll_opamp.h

Data Fields

- ***uint32_t PowerMode***
- ***uint32_t FunctionalMode***
- ***uint32_t InputNonInverting***
- ***uint32_t InputInverting***

Field Documentation

- ***uint32_t LL_OPAMP_InitTypeDef::PowerMode***
  Set the power mode Normal or Low-Power and the speed mode Normal or High. This parameter can be a value of ***OPAMP_LL_EC_POWERMODE*** This feature can be modified afterwards using unitary function **LL_OPAMP_SetPowerMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::FunctionalMode***
  Set OPAMP functional mode by setting internal connections: OPAMP operation in standalone, follower, ... This parameter can be a value of ***OPAMP_LL_EC_FUNCTIONAL_MODE***
  Note:
  – If OPAMP is configured in mode PGA, the gain can be configured using function **LL_OPAMP_SetPGAGain()**. This feature can be modified afterwards using unitary function **LL_OPAMP_SetFunctionalMode()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputNonInverting***
  Set OPAMP input non-inverting connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_NONINVERTING*** This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputNonInverting()**.
- ***uint32_t LL_OPAMP_InitTypeDef::InputInverting***
  Set OPAMP inverting input connection. This parameter can be a value of ***OPAMP_LL_EC_INPUT_INVERTING***
  Note:
  – OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin), this parameter is discarded. This feature can be modified afterwards using unitary function **LL_OPAMP_SetInputInverting()**.

111.2 OPAMP Firmware driver API description

The following section lists the various functions of the OPAMP library.

111.2.1 Detailed description of functions

LL_OPAMP_SetCommonPowerRange

Function name

__STATIC_INLINE void LL_OPAMP_SetCommonPowerRange (OPAMP_Common_TypeDef * OPAMPxy_COMMON, uint32_t PowerRange)

Function description

Set OPAMP power range.

Parameters

- **OPAMPxy_COMMON:** OPAMP common instance
- **PowerRange:** This parameter can be one of the following values:
  – LL_OPAMP_POWERSUPPLY_RANGE_LOW
  – LL_OPAMP_POWERSUPPLY_RANGE_HIGH

Return values

- **None:**

Notes

- The OPAMP power range applies to several OPAMP instances (if several OPAMP instances available on the selected device). It is mandatory to set already the power range to high for performance purpose.
- On this STM32 series, setting of this feature is conditioned to OPAMP state: All OPAMP instances of the OPAMP common group must be disabled. This check can be done with function LL_OPAMP_IsEnabled() for each OPAMP instance.
- LL_OPAMP_POWERSUPPLY_RANGE_HIGH parameter is highly recommended for performance purpose.

Reference Manual to LL API cross reference:

- CSR OPARANGE LL_OPAMP_SetCommonPowerRange

LL_OPAMP_GetCommonPowerRange

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetCommonPowerRange (const OPAMP_Common_TypeDef * OPAMPxy_COMMON)

Function description

Get OPAMP power range.

Parameters

- **OPAMPxy_COMMON:** OPAMP common instance

Return values

- **Returned:** value can be one of the following values:
  – LL_OPAMP_POWERSUPPLY_RANGE_LOW
  – LL_OPAMP_POWERSUPPLY_RANGE_HIGH

Notes

- The OPAMP power range applies to several OPAMP instances (if several OPAMP instances available on the selected device).

Reference Manual to LL API cross reference:

- CSR OPARANGE LL_OPAMP_GetCommonPowerRange

LL_OPAMP_SetPowerMode

Function name

__STATIC_INLINE void LL_OPAMP_SetPowerMode (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode)

Function description

Set OPAMP power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_HIGHSPEED

Return values

- **None:**

Notes

- The OPAMP must be disabled to change this configuration.

Reference Manual to LL API cross reference:

- CSR OPALPM LL_OPAMP_SetPowerMode

LL_OPAMP_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPowerMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP power mode.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_HIGHSPEED

Reference Manual to LL API cross reference:

- CSR OPALPM & HSM LL_OPAMP_GetPowerMode

LL_OPAMP_SetMode

Function name

__STATIC_INLINE void LL_OPAMP_SetMode (OPAMP_TypeDef * OPAMPx, uint32_t Mode)

Function description

Set OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance
- **Mode:** This parameter can be one of the following values:
  - LL_OPAMP_MODE_FUNCTIONAL
  - LL_OPAMP_MODE_CALIBRATION

Return values

- **None:**

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.
- On this STM32 series, during calibration, OPAMP functional mode must be set to standalone or follower mode (in order to open internal connections to resistors of PGA mode). Refer to function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_SetMode

LL_OPAMP_GetMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP mode calibration or functional.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  – LL_OPAMP_MODE_FUNCTIONAL
  – LL_OPAMP_MODE_CALIBRATION

Notes

- OPAMP mode corresponds to functional or calibration mode: functional mode: OPAMP operation in standalone, follower, ... Set functional mode using function LL_OPAMP_SetFunctionalMode().calibration mode: offset calibration of the selected transistors differential pair NMOS or PMOS.

Reference Manual to LL API cross reference:

- CSR CALON LL_OPAMP_GetMode

LL_OPAMP_SetFunctionalMode

Function name

__STATIC_INLINE void LL_OPAMP_SetFunctionalMode (OPAMP_TypeDef * OPAMPx, uint32_t FunctionalMode)

Function description

Set OPAMP functional mode by setting internal connections.

Parameters

- **OPAMPx:** OPAMP instance
- **FunctionalMode:** This parameter can be one of the following values:
  – LL_OPAMP_MODE_STANDALONE
  – LL_OPAMP_MODE_FOLLOWER
  – LL_OPAMP_MODE_PGA

Return values

- **None:**

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.

Reference Manual to LL API cross reference:

- CSR OPAMODE LL_OPAMP_SetFunctionalMode

LL_OPAMP_GetFunctionalMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetFunctionalMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP functional mode from setting of internal connections.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  – LL_OPAMP_MODE_STANDALONE
  – LL_OPAMP_MODE_FOLLOWER
  – LL_OPAMP_MODE_PGA

Reference Manual to LL API cross reference:

- CSR OPAMODE LL_OPAMP_GetFunctionalMode

LL_OPAMP_SetPGAGain

Function name

__STATIC_INLINE void LL_OPAMP_SetPGAGain (OPAMP_TypeDef * OPAMPx, uint32_t PGAGain)

Function description

Set OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance
- **PGAGain:** This parameter can be one of the following values:
  – LL_OPAMP_PGA_GAIN_2
  – LL_OPAMP_PGA_GAIN_4
  – LL_OPAMP_PGA_GAIN_8
  – LL_OPAMP_PGA_GAIN_16

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_SetPGAGain

LL_OPAMP_GetPGAGain

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetPGAGain (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP PGA gain.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_PGA_GAIN_2
  - LL_OPAMP_PGA_GAIN_4
  - LL_OPAMP_PGA_GAIN_8
  - LL_OPAMP_PGA_GAIN_16

Notes

- Preliminarily, OPAMP must be set in mode PGA using function LL_OPAMP_SetFunctionalMode().

Reference Manual to LL API cross reference:

- CSR PGGAIN LL_OPAMP_GetPGAGain

LL_OPAMP_SetInputNonInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputNonInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputNonInverting)

Function description

Set OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputNonInverting:** This parameter can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINV_DAC1_CH1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_SetInputNonInverting

LL_OPAMP_GetInputNonInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputNonInverting (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP non-inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_INPUT_NONINVERT_IO0
  - LL_OPAMP_INPUT_NONINV_DAC1_CH1

Reference Manual to LL API cross reference:

- CSR VPSEL LL_OPAMP_GetInputNonInverting

LL_OPAMP_SetInputInverting

Function name

__STATIC_INLINE void LL_OPAMP_SetInputInverting (OPAMP_TypeDef * OPAMPx, uint32_t InputInverting)

Function description

Set OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance
- **InputInverting:** This parameter can be one of the following values:
  – LL_OPAMP_INPUT_INVERT_IO0
  – LL_OPAMP_INPUT_INVERT_IO1
  – LL_OPAMP_INPUT_INVERT_CONNECT_NO

Return values

- **None:**

Notes

- OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_SetInputInverting

LL_OPAMP_GetInputInverting

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetInputInverting (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP inverting input connection.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  – LL_OPAMP_INPUT_INVERT_IO0
  – LL_OPAMP_INPUT_INVERT_IO1
  – LL_OPAMP_INPUT_INVERT_CONNECT_NO

Reference Manual to LL API cross reference:

- CSR VMSEL LL_OPAMP_GetInputInverting

LL_OPAMP_SetNonInvertingInput

Function name

__STATIC_INLINE void LL_OPAMP_SetNonInvertingInput (OPAMP_TypeDef * OPAMPx, uint32_t NonInvertingInput)

Function description

LL_OPAMP_SetInvertingInput

Function name

__STATIC_INLINE void LL_OPAMP_SetInvertingInput (OPAMP_TypeDef * OPAMPx, uint32_t InvertingInput)

Function description

LL_OPAMP_SetTrimmingMode

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingMode (OPAMP_TypeDef * OPAMPx, uint32_t TrimmingMode)

Function description

Set OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance
- **TrimmingMode:** This parameter can be one of the following values:
  – LL_OPAMP_TRIMMING_FACTORY
  – LL_OPAMP_TRIMMING_USER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_SetTrimmingMode

LL_OPAMP_GetTrimmingMode

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingMode (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP trimming mode.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  – LL_OPAMP_TRIMMING_FACTORY
  – LL_OPAMP_TRIMMING_USER

Reference Manual to LL API cross reference:

- CSR USERTRIM LL_OPAMP_GetTrimmingMode

LL_OPAMP_SetCalibrationSelection

Function name

__STATIC_INLINE void LL_OPAMP_SetCalibrationSelection (OPAMP_TypeDef * OPAMPx, uint32_t TransistorsDiffPair)

Function description

Set OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS

Return values

- **None:**

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_SetCalibrationSelection

LL_OPAMP_GetCalibrationSelection

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetCalibrationSelection (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP offset to calibrate the selected transistors differential pair NMOS or PMOS.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **Returned:** value can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS

Notes

- Preliminarily, OPAMP must be set in mode calibration using function LL_OPAMP_SetMode().

Reference Manual to LL API cross reference:

- CSR CALSEL LL_OPAMP_GetCalibrationSelection

LL_OPAMP_IsCalibrationOutputSet

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsCalibrationOutputSet (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP calibration result of toggling output.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Notes

- This functions returns: 0 if OPAMP calibration output is reset 1 if OPAMP calibration output is set

Reference Manual to LL API cross reference:

- CSR CALOUT LL_OPAMP_IsCalibrationOutputSet

LL_OPAMP_SetTrimmingValue

Function name

__STATIC_INLINE void LL_OPAMP_SetTrimmingValue (OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair, uint32_t TrimmingValue)

Function description

Set OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_HIGHSPEED
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS
- **TrimmingValue:** 0x00...0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- OTR TRIMOFFSETN LL_OPAMP_SetTrimmingValue
- OTR TRIMOFFSETP LL_OPAMP_SetTrimmingValue
- LPOTR TRIMLPOFFSETN LL_OPAMP_SetTrimmingValue
- LPOTR TRIMLPOFFSETP LL_OPAMP_SetTrimmingValue

LL_OPAMP_GetTrimmingValue

Function name

__STATIC_INLINE uint32_t LL_OPAMP_GetTrimmingValue (const OPAMP_TypeDef * OPAMPx, uint32_t PowerMode, uint32_t TransistorsDiffPair)

Function description

Get OPAMP trimming factor for the selected transistors differential pair NMOS or PMOS, corresponding to the selected power mode.

Parameters

- **OPAMPx:** OPAMP instance
- **PowerMode:** This parameter can be one of the following values:
  - LL_OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_NORMALSPEED
  - LL_OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED
  - LL_OPAMP_POWERMODE_LOWPOWER_HIGHSPEED
- **TransistorsDiffPair:** This parameter can be one of the following values:
  - LL_OPAMP_TRIMMING_NMOS
  - LL_OPAMP_TRIMMING_PMOS

Return values

- **0x0...0x1F:**

Reference Manual to LL API cross reference:

- OTR TRIMOFFSETN LL_OPAMP_GetTrimmingValue
- OTR TRIMOFFSETP LL_OPAMP_GetTrimmingValue
- LPOTR TRIMLPOFFSETN LL_OPAMP_GetTrimmingValue
- LPOTR TRIMLPOFFSETP LL_OPAMP_GetTrimmingValue

LL_OPAMP_Enable

Function name

__STATIC_INLINE void LL_OPAMP_Enable (OPAMP_TypeDef * OPAMPx)

Function description

Enable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Notes

- After enable from off state, OPAMP requires a delay to fulfill wake up time specification. Refer to device datasheet, parameter "tWAKEUP".

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Enable

LL_OPAMP_Disable

Function name

__STATIC_INLINE void LL_OPAMP_Disable (OPAMP_TypeDef * OPAMPx)

Function description

Disable OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_Disable

LL_OPAMP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_OPAMP_IsEnabled (const OPAMP_TypeDef * OPAMPx)

Function description

Get OPAMP instance enable state (0: OPAMP is disabled, 1: OPAMP is enabled)

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OPAMPXEN LL_OPAMP_IsEnabled

LL_OPAMP_DeInit

Function name

ErrorStatus LL_OPAMP_DeInit (OPAMP_TypeDef * OPAMPx)

Function description

De-initialize registers of the selected OPAMP instance to their default reset values.

Parameters

- **OPAMPx:** OPAMP instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: OPAMP registers are de-initialized
  – ERROR: OPAMP registers are not de-initialized

LL_OPAMP_Init

Function name

ErrorStatus LL_OPAMP_Init (OPAMP_TypeDef * OPAMPx, const LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Initialize some features of OPAMP instance.

Parameters

- **OPAMPx:** OPAMP instance
- **OPAMP_InitStruct:** Pointer to a LL_OPAMP_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: OPAMP registers are initialized
  – ERROR: OPAMP registers are not initialized

Notes

- This function reset bit of calibration mode to ensure to be in functional mode, in order to have OPAMP parameters (inputs selection, ...) set with the corresponding OPAMP mode to be effective.
- This function configures features of the selected OPAMP instance. Some features are also available at scope OPAMP common instance (common to several OPAMP instances).

LL_OPAMP_StructInit

Function name

void LL_OPAMP_StructInit (LL_OPAMP_InitTypeDef * OPAMP_InitStruct)

Function description

Set each LL_OPAMP_InitTypeDef field to default value.

Parameters

- **OPAMP_InitStruct:** pointer to a LL_OPAMP_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

111.3 OPAMP Firmware driver defines

The following section lists the various define and macros of the module.

111.3.1 OPAMP

OPAMP

OPAMP functional mode

LL_OPAMP_MODE_STANDALONE

OPAMP functional mode, OPAMP operation in standalone

LL_OPAMP_MODE_FOLLOWER

OPAMP functional mode, OPAMP operation in follower

LL_OPAMP_MODE_PGA

OPAMP functional mode, OPAMP operation in PGA

Definitions of OPAMP hardware constraints delays

LL_OPAMP_DELAY_STARTUP_US

Delay for OPAMP startup time

OPAMP input inverting

LL_OPAMP_INPUT_INVERT_IO0

OPAMP inverting input connected to GPIO pin (valid also in PGA mode for filtering). Note: OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

LL_OPAMP_INPUT_INVERT_IO1

OPAMP inverting input (low leakage input) connected to GPIO pin (available only on package BGA132). Note: OPAMP inverting input is used with OPAMP in mode standalone or PGA with external capacitors for filtering circuit. Otherwise (OPAMP in mode follower), OPAMP inverting input is not used (not connected to GPIO pin).

LL_OPAMP_INPUT_INVERT_CONNECT_NO

OPAMP inverting input not externally connected (intended for OPAMP in mode follower or PGA without external capacitors for filtering)

OPAMP inputs legacy literals name

LL_OPAMP_NONINVERTINGINPUT_IO0

LL_OPAMP_NONINVERTINGINPUT_DAC_CH

LL_OPAMP_INVERTINGINPUT_IO0

LL_OPAMP_INVERTINGINPUT_IO1

LL_OPAMP_INVERTINGINPUT_CONNECT_NO

LL_OPAMP_INPUT_NONINVERT_DAC1_CH1

OPAMP input non-inverting

LL_OPAMP_INPUT_NONINVERT_IO0

OPAMP non inverting input connected to GPIO pin (pin PA0 for OPAMP1, pin PA6 for OPAMP2)

LL_OPAMP_INPUT_NONINV_DAC1_CH1

OPAMP non inverting input connected to DAC1 channel output(channel1 for OPAMP1, channel2 for OPAMP2)

OPAMP mode calibration or functional.

LL_OPAMP_MODE_FUNCTIONAL

OPAMP functional mode

LL_OPAMP_MODE_CALIBRATION

OPAMP calibration mode

OPAMP PGA gain (relevant when OPAMP is in functional mode PGA)

LL_OPAMP_PGA_GAIN_2

OPAMP PGA gain 2

LL_OPAMP_PGA_GAIN_4

OPAMP PGA gain 4

LL_OPAMP_PGA_GAIN_8

OPAMP PGA gain 8

LL_OPAMP_PGA_GAIN_16

OPAMP PGA gain 16

OPAMP power mode

LL_OPAMP_POWERMODE_NORMALPOWER_NORMALSPEED

OPAMP power mode normal speed normal

LL_OPAMP_POWERMODE_LOWPOWER_NORMALSPEED

OPAMP power mode low-power speed normal

LL_OPAMP_POWERMODE_NORMALPOWER_HIGHSPEED

OPAMP power mode normal speed high

LL_OPAMP_POWERMODE_LOWPOWER_HIGHSPEED

OPAMP power mode low-power speed high

OPAMP power supply range

LL_OPAMP_POWERSUPPLY_RANGE_LOW

Power supply range low

LL_OPAMP_POWERSUPPLY_RANGE_HIGH

Power supply range high - recommended for performance purpose

OPAMP trimming mode

LL_OPAMP_TRIMMING_FACTORY

OPAMP trimming factors set to factory values

LL_OPAMP_TRIMMING_USER

OPAMP trimming factors set to user values

OPAMP trimming of transistors differential pair NMOS or PMOS

LL_OPAMP_TRIMMING_NMOS

OPAMP trimming of transistors differential pair NMOS

LL_OPAMP_TRIMMING_PMOS

OPAMP trimming of transistors differential pair PMOS

OPAMP helper macro

__LL_OPAMP_COMMON_INSTANCE

Description:
- Helper macro to select the OPAMP common instance to which is belonging the selected OPAMP instance.

Parameters:
- __OPAMPx__: OPAMP instance

Return value:
- OPAMP: common instance

Notes:
- OPAMP common register instance can be used to set parameters common to several OPAMP instances. Refer to functions having argument "OPAMPxy_COMMON" as parameter.

__LL_OPAMP_IS_ENABLED_ALL_COMMON_INSTANCE

Description:
- Helper macro to check if all OPAMP instances sharing the same OPAMP common instance are disabled.

Return value:
- 0: All OPAMP instances sharing the same OPAMP common instance are disabled. 1: At least one OPAMP instance sharing the same OPAMP common instance is enabled

Notes:
- This check is required by functions with setting conditioned to OPAMP state: All OPAMP instances of the OPAMP common group must be disabled. Refer to functions having argument "OPAMPxy_COMMON" as parameter.

Common write and read registers macro

LL_OPAMP_WriteReg

Description:
- Write a value in OPAMP register.

Parameters:
- __INSTANCE__: OPAMP Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_OPAMP_ReadReg

Description:
- Read a value in OPAMP register.

Parameters:
- __INSTANCE__: OPAMP Instance
- __REG__: Register to be read

Return value:
- Register: value

112 LL PKA Generic Driver

112.1 PKA Firmware driver registers structures

112.1.1 LL_PKA_InitTypeDef

LL_PKA_InitTypeDef is defined in the stm32u5xx_ll_pka.h

Data Fields

- ***uint32_t Mode***

Field Documentation

- ***uint32_t LL_PKA_InitTypeDef::Mode***
  Specifies the PKA operation mode. This parameter can be a value of ***PKA_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_PKA_SetMode()**.

112.2 PKA Firmware driver API description

The following section lists the various functions of the PKA library.

112.2.1 Detailed description of functions

LL_PKA_Config

Function name

__STATIC_INLINE void LL_PKA_Config (PKA_TypeDef * PKAx, uint32_t Mode)

Function description

Configure PKA peripheral.

Parameters

- **PKAx:** PKA Instance.
- **Mode:** This parameter can be one of the following values:
  - LL_PKA_MODE_MONTGOMERY_PARAM
  - LL_PKA_MODE_MODULAR_EXP
  - LL_PKA_MODE_ECDSA_SIGNATURE
  - LL_PKA_MODE_ECDSA_VERIFICATION
  - LL_PKA_MODE_POINT_CHECK
  - LL_PKA_MODE_RSA_CRT_EXP
  - LL_PKA_MODE_MODULAR_INV
  - LL_PKA_MODE_ARITHMETIC_ADD
  - LL_PKA_MODE_ARITHMETIC_SUB
  - LL_PKA_MODE_ARITHMETIC_MUL
  - LL_PKA_MODE_COMPARISON
  - LL_PKA_MODE_MODULAR_REDUC
  - LL_PKA_MODE_MODULAR_ADD
  - LL_PKA_MODE_MODULAR_SUB
  - LL_PKA_MODE_MONTGOMERY_MUL
  - LL_PKA_MODE_MODULAR_EXP_PROTECT
  - LL_PKA_MODE_DOUBLE_BASE_LADDER
  - LL_PKA_MODE_ECC_PROJECTIVE_AFF
  - LL_PKA_MODE_ECC_COMPLETE_ADD
  - LL_PKA_MODE_ECC_MUL
  - LL_PKA_MODE_MODULAR_EXP_FAST

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_Config

LL_PKA_Enable

Function name

__STATIC_INLINE void LL_PKA_Enable (PKA_TypeDef * PKAx)

Function description

Enable PKA peripheral.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_PKA_Enable

LL_PKA_Disable

Function name

__STATIC_INLINE void LL_PKA_Disable (PKA_TypeDef * PKAx)

Function description

Disable PKA peripheral.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR EN LL_PKA_Disable

LL_PKA_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabled (const PKA_TypeDef * PKAx)

Function description

Check if the PKA peripheral is enabled or disabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR EN LL_PKA_IsEnabled

LL_PKA_SetMode

Function name

__STATIC_INLINE void LL_PKA_SetMode (PKA_TypeDef * PKAx, uint32_t Mode)

Function description

Set PKA operating mode.

Parameters

- **PKAx:** PKA Instance.
- **Mode:** This parameter can be one of the following values:
  – LL_PKA_MODE_MONTGOMERY_PARAM
  – LL_PKA_MODE_MODULAR_EXP
  – LL_PKA_MODE_ECDSA_SIGNATURE
  – LL_PKA_MODE_ECDSA_VERIFICATION
  – LL_PKA_MODE_POINT_CHECK
  – LL_PKA_MODE_RSA_CRT_EXP
  – LL_PKA_MODE_MODULAR_INV
  – LL_PKA_MODE_ARITHMETIC_ADD
  – LL_PKA_MODE_ARITHMETIC_SUB
  – LL_PKA_MODE_ARITHMETIC_MUL
  – LL_PKA_MODE_COMPARISON
  – LL_PKA_MODE_MODULAR_REDUC
  – LL_PKA_MODE_MODULAR_ADD
  – LL_PKA_MODE_MODULAR_SUB
  – LL_PKA_MODE_MONTGOMERY_MUL
  – LL_PKA_MODE_MODULAR_EXP_PROTECT
  – LL_PKA_MODE_DOUBLE_BASE_LADDER
  – LL_PKA_MODE_ECC_PROJECTIVE_AFF
  – LL_PKA_MODE_ECC_COMPLETE_ADD
  – LL_PKA_MODE_ECC_MUL
  – LL_PKA_MODE_MODULAR_EXP_FAST

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_SetMode

LL_PKA_GetMode

Function name

__STATIC_INLINE uint32_t LL_PKA_GetMode (const PKA_TypeDef * PKAx)

Function description

Get PKA operating mode.

Parameters

- **PKAx:** PKA Instance.

Return values

- **Returned:** value can be one of the following values:
  – LL_PKA_MODE_MONTGOMERY_PARAM
  – LL_PKA_MODE_MODULAR_EXP
  – LL_PKA_MODE_ECDSA_SIGNATURE
  – LL_PKA_MODE_ECDSA_VERIFICATION
  – LL_PKA_MODE_POINT_CHECK
  – LL_PKA_MODE_RSA_CRT_EXP
  – LL_PKA_MODE_MODULAR_INV
  – LL_PKA_MODE_ARITHMETIC_ADD
  – LL_PKA_MODE_ARITHMETIC_SUB
  – LL_PKA_MODE_ARITHMETIC_MUL
  – LL_PKA_MODE_COMPARISON
  – LL_PKA_MODE_MODULAR_REDUC
  – LL_PKA_MODE_MODULAR_ADD
  – LL_PKA_MODE_MODULAR_SUB
  – LL_PKA_MODE_MONTGOMERY_MUL
  – LL_PKA_MODE_MODULAR_EXP_PROTECT
  – LL_PKA_MODE_DOUBLE_BASE_LADDER
  – LL_PKA_MODE_ECC_PROJECTIVE_AFF
  – LL_PKA_MODE_ECC_COMPLETE_ADD
  – LL_PKA_MODE_ECC_MUL
  – LL_PKA_MODE_MODULAR_EXP_FAST

Reference Manual to LL API cross reference:

- CR MODE LL_PKA_GetMode

LL_PKA_Start

Function name

__STATIC_INLINE void LL_PKA_Start (PKA_TypeDef * PKAx)

Function description

Start the operation selected using LL_PKA_SetMode.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR START LL_PKA_Start

LL_PKA_EnableIT_ADDRERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_ADDRERR (PKA_TypeDef * PKAx)

Function description

Enable address error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_EnableIT_ADDRERR

LL_PKA_EnableIT_RAMERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_RAMERR (PKA_TypeDef * PKAx)

Function description

Enable RAM error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_EnableIT_RAMERR

LL_PKA_EnableIT_OPERR

Function name

__STATIC_INLINE void LL_PKA_EnableIT_OPERR (PKA_TypeDef * PKAx)

Function description

Enable OPERATION error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_EnableIT_OPERR

LL_PKA_EnableIT_PROCEND

Function name

__STATIC_INLINE void LL_PKA_EnableIT_PROCEND (PKA_TypeDef * PKAx)

Function description

Enable end of operation interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_EnableIT_PROCEND

LL_PKA_DisableIT_ADDERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_ADDERR (PKA_TypeDef * PKAx)

Function description

Disable address error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_DisableIT_ADDERR

LL_PKA_DisableIT_RAMERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_RAMERR (PKA_TypeDef * PKAx)

Function description

Disable RAM error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_DisableIT_RAMERR

LL_PKA_DisableIT_PROCEND

Function name

__STATIC_INLINE void LL_PKA_DisableIT_PROCEND (PKA_TypeDef * PKAx)

Function description

Disable End of operation interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_DisableIT_PROCEND

LL_PKA_DisableIT_OPERR

Function name

__STATIC_INLINE void LL_PKA_DisableIT_OPERR (PKA_TypeDef * PKAx)

Function description

Disable OPERATION error interrupt.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_EnableIT_OPERR

LL_PKA_IsEnabledIT_ADDRERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_ADDRERR (const PKA_TypeDef * PKAx)

Function description

Check if address error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ADDRERRIE LL_PKA_IsEnabledIT_ADDRERR

LL_PKA_IsEnabledIT_RAMERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_RAMERR (const PKA_TypeDef * PKAx)

Function description

Check if RAM error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RAMERRIE LL_PKA_IsEnabledIT_RAMERR

LL_PKA_IsEnabledIT_OPERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_OPERR (const PKA_TypeDef * PKAx)

Function description

Check if OPERATION error interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR OPERRIE LL_PKA_IsEnabledIT_OPERR

LL_PKA_IsEnabledIT_PROCEND

Function name

__STATIC_INLINE uint32_t LL_PKA_IsEnabledIT_PROCEND (const PKA_TypeDef * PKAx)

Function description

Check if end of operation interrupt is enabled.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PROCENDIE LL_PKA_IsEnabledIT_PROCEND

LL_PKA_IsActiveFlag_ADDRERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_ADDRERR (const PKA_TypeDef * PKAx)

Function description

Get PKA address error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR ADDRERRF LL_PKA_IsActiveFlag_ADDRERR

LL_PKA_IsActiveFlag_RAMERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_RAMERR (const PKA_TypeDef * PKAx)

Function description

Get PKA RAM error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RAMERRF LL_PKA_IsActiveFlag_RAMERR

LL_PKA_IsActiveFlag_OPERR

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_OPERR (const PKA_TypeDef * PKAx)

Function description

Get PKA OPERATION error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OPERRF LL_PKA_IsActiveFlag_OPERR

LL_PKA_IsActiveFlag_PROCEND

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_PROCEND (const PKA_TypeDef * PKAx)

Function description

Get PKA end of operation flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR PROCENDF LL_PKA_IsActiveFlag_PROCEND

LL_PKA_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_PKA_IsActiveFlag_BUSY (const PKA_TypeDef * PKAx)

Function description

Get PKA busy flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BUSY LL_PKA_IsActiveFlag_BUSY

LL_PKA_ClearFlag_ADDERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_ADDERR (PKA_TypeDef * PKAx)

Function description

Clear PKA address error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR ADDRERRFC LL_PKA_ClearFlag_ADDERR

LL_PKA_ClearFlag_RAMERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_RAMERR (PKA_TypeDef * PKAx)

Function description

Clear PKA RAM error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR RAMERRFC LL_PKA_ClearFlag_RAMERR

LL_PKA_ClearFlag_OPERR

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_OPERR (PKA_TypeDef * PKAx)

Function description

Clear PKA OPERATION error flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR OPERRFC LL_PKA_ClearFlag_OPERR

LL_PKA_ClearFlag_PROCEND

Function name

__STATIC_INLINE void LL_PKA_ClearFlag_PROCEND (PKA_TypeDef * PKAx)

Function description

Clear PKA end of operation flag.

Parameters

- **PKAx:** PKA Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CLRFR PROCENDFC LL_PKA_ClearFlag_PROCEND

LL_PKA_DeInit

Function name

ErrorStatus LL_PKA_DeInit (const PKA_TypeDef * PKAx)

Function description

De-initialize PKA registers (Registers restored to their default values).

Parameters

- **PKAx:** PKA Instance.

Return values

- **ErrorStatus:**
  – SUCCESS: PKA registers are de-initialized
  – ERROR: PKA registers are not de-initialized

LL_PKA_Init

Function name

ErrorStatus LL_PKA_Init (PKA_TypeDef * PKAx, LL_PKA_InitTypeDef * PKA_InitStruct)

Function description

Initialize PKA registers according to the specified parameters in PKA_InitStruct.

Parameters

- **PKAx:** PKA Instance.
- **PKA_InitStruct:** pointer to a LL_PKA_InitTypeDef structure that contains the configuration information for the specified PKA peripheral.

Return values

- **ErrorStatus:**
  – SUCCESS: PKA registers are initialized according to PKA_InitStruct content
  – ERROR: Not applicable

LL_PKA_StructInit

Function name

void LL_PKA_StructInit (LL_PKA_InitTypeDef * PKA_InitStruct)

Function description

Set each LL_PKA_InitTypeDef field to default value.

Parameters

- **PKA_InitStruct:** pointer to a LL_PKA_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

112.3 PKA Firmware driver defines

The following section lists the various define and macros of the module.

112.3.1 PKA

PKA

Get Flags Defines

LL_PKA_SR_ADDRERRF

LL_PKA_SR_RAMERRF

LL_PKA_SR_PROCENDF

LL_PKA_SR_BUSY

LL_PKA_SR_INITOK

LL_PKA_SR_OPERRF

IT Defines

LL_PKA_CR_ADDRERRIE

LL_PKA_CR_RAMERRIE

LL_PKA_CR_PROCENDIE

LL_PKA_CLRFR_PROCENDFC

LL_PKA_CLRFR_RAMERRFC

LL_PKA_CLRFR_ADDRERRFC

LL_PKA_CR_OPERRIE

LL_PKA_CLRFR_OPERRFC

Operation Mode

LL_PKA_MODE_MODULAR_EXP

modular exponentiation

LL_PKA_MODE_MONTGOMERY_PARAM

Compute Montgomery parameter only

LL_PKA_MODE_MODULAR_EXP_FAST

modular exponentiation fast mode

LL_PKA_MODE_MODULAR_EXP_PROTECT

modular exponentiation protect mode

LL_PKA_MODE_ECC_MUL

compute ECC kP operation

LL_PKA_MODE_ECC_COMPLETE_ADD

ECC complete addition

LL_PKA_MODE_ECDSA_SIGNATURE

ECDSA signature

LL_PKA_MODE_ECDSA_VERIFICATION

ECDSA verification

LL_PKA_MODE_POINT_CHECK

Point check

LL_PKA_MODE_RSA_CRT_EXP

RSA CRT exponentiation

LL_PKA_MODE_MODULAR_INV

Modular inversion

LL_PKA_MODE_ARITHMETIC_ADD

Arithmetic addition

LL_PKA_MODE_ARITHMETIC_SUB

Arithmetic subtraction

LL_PKA_MODE_ARITHMETIC_MUL

Arithmetic multiplication

LL_PKA_MODE_COMPARISON

Comparison

LL_PKA_MODE_MODULAR_REDUC

Modular reduction

LL_PKA_MODE_MODULAR_ADD

Modular addition

LL_PKA_MODE_MODULAR_SUB

Modular subtraction

LL_PKA_MODE_MONTGOMERY_MUL

Montgomery multiplication

LL_PKA_MODE_DOUBLE_BASE_LADDER

Double base ladder

LL_PKA_MODE_ECC_PROJECTIVE_AFF

ECC projective to affine

Common Write and read registers Macros

LL_PKA_WriteReg

Description:
- Write a value in PKA register.

Parameters:
- __INSTANCE__: PKA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_PKA_ReadReg

Description:

- Read a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be read

Return value:

- Register: value

113 LL PWR Generic Driver

113.1 PWR Firmware driver API description

The following section lists the various functions of the PWR library.

113.1.1 Detailed description of functions

LL_PWR_SetPowerMode

Function name

__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t Mode)

Function description

Set system power mode.

Parameters

- **Mode:** : This parameter can be one of the following values:
  – LL_PWR_STOP0_MODE
  – LL_PWR_STOP1_MODE
  – LL_PWR_STOP2_MODE
  – LL_PWR_STOP3_MODE
  – LL_PWR_STANDBY_MODE
  – LL_PWR_SHUTDOWN_MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 LPMS LL_PWR_SetPowerMode

LL_PWR_GetPowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPowerMode (void)

Function description

Get system power mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_STOP0_MODE
  – LL_PWR_STOP1_MODE
  – LL_PWR_STOP2_MODE
  – LL_PWR_STOP3_MODE
  – LL_PWR_STANDBY_MODE
  – LL_PWR_SHUTDOWN_MODE

Reference Manual to LL API cross reference:

- CR1 LPMS LL_PWR_GetPowerMode

LL_PWR_SetSRAM2SBRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM2SBRetention (uint32_t SRAM2PageRetention)

Function description

Set the SRAM2 page(s) retention in Standby mode.

Parameters

- **SRAM2PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM2_SB_NO_RETENTION
  – LL_PWR_SRAM2_SB_PAGE1_RETENTION
  – LL_PWR_SRAM2_SB_PAGE2_RETENTION
  – LL_PWR_SRAM2_SB_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RRSB1 LL_PWR_SetSRAM2SBRetention
-
- CR1 RRSB2 LL_PWR_SetSRAM2SBRetention

LL_PWR_GetSRAM2SBRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM2SBRetention (void)

Function description

Get the SRAM2 page(s) retention in Standby mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM2_SB_NO_RETENTION
  – LL_PWR_SRAM2_SB_PAGE1_RETENTION
  – LL_PWR_SRAM2_SB_PAGE2_RETENTION
  – LL_PWR_SRAM2_SB_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 RRSB1 LL_PWR_GetSRAM2SBRetention
-
- CR1 RRSB2 LL_PWR_GetSRAM2SBRetention

LL_PWR_EnableUltraLowPowerMode

Function name

__STATIC_INLINE void LL_PWR_EnableUltraLowPowerMode (void)

Function description

Enable BOR ultra low power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UPLMEN LL_PWR_EnableUltraLowPowerMode

LL_PWR_DisableUltraLowPowerMode

Function name

__STATIC_INLINE void LL_PWR_DisableUltraLowPowerMode (void)

Function description

Disable BOR ultra low-power mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UPLMEN LL_PWR_DisableUltraLowPowerMode

LL_PWR_IsEnabledUltraLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUltraLowPowerMode (void)

Function description

Check if BOR ultra low power mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UPLMEN LL_PWR_IsEnabledUltraLowPowerMode

LL_PWR_SetSRAM1RunRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM1RunRetention (uint32_t SRAM1Retention)

Function description

Set the SRAM1 retention in Run mode.

Parameters

- **SRAM1Retention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM1_RUN_NO_RETENTION
  - LL_PWR_SRAM1_RUN_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SRAM1PD LL_PWR_SetSRAM1RunRetention
-

LL_PWR_GetSRAM1RunRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM1RunRetention (void)

Function description

Get the SRAM1 retention in Run mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM1_RUN_NO_RETENTION
  - LL_PWR_SRAM1_RUN_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 SRAM1PD LL_PWR_GetSRAM1RunRetention
-

LL_PWR_SetSRAM2RunRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM2RunRetention (uint32_t SRAM2Retention)

Function description

Set the SRAM2 retention in Run mode.

Parameters

- **SRAM2Retention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM2_RUN_NO_RETENTION
  - LL_PWR_SRAM2_RUN_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SRAM2PD LL_PWR_SetSRAM2RunRetention
-

LL_PWR_GetSRAM2RunRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM2RunRetention (void)

Function description

Get the SRAM2 retention in Run mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM2_RUN_NO_RETENTION
  - LL_PWR_SRAM2_RUN_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 SRAM2PD LL_PWR_GetSRAM2RunRetention
-

LL_PWR_SetSRAM3RunRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM3RunRetention (uint32_t SRAM3Retention)

Function description

Set the SRAM3 retention in Run mode.

Parameters

- **SRAM3Retention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM3_RUN_NO_RETENTION
  - LL_PWR_SRAM3_RUN_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SRAM3PD LL_PWR_SetSRAM3RunRetention
-

LL_PWR_GetSRAM3RunRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM3RunRetention (void)

Function description

Get the SRAM3 retention in Run mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM3_RUN_NO_RETENTION
  - LL_PWR_SRAM3_RUN_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 SRAM3PD LL_PWR_GetSRAM3RunRetention
-

LL_PWR_SetSRAM4RunRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM4RunRetention (uint32_t SRAM4Retention)

Function description

Set the SRAM4 retention in Run mode.

Parameters

- **SRAM4Retention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM4_RUN_NO_RETENTION
  - LL_PWR_SRAM4_RUN_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SRAM4PD LL_PWR_SetSRAM4RunRetention
-

LL_PWR_GetSRAM4RunRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM4RunRetention (void)

Function description

Get the SRAM4 retention in Run mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM4_RUN_NO_RETENTION
  – LL_PWR_SRAM4_RUN_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 SRAM4PD LL_PWR_GetSRAM4RunRetention
-

LL_PWR_SetSRAM5RunRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM5RunRetention (uint32_t SRAM5Retention)

Function description

Set the SRAM5 retention in Run mode.

Parameters

- **SRAM5Retention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM5_RUN_NO_RETENTION
  – LL_PWR_SRAM5_RUN_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SRAM5PD LL_PWR_SetSRAM5RunRetention
-

LL_PWR_GetSRAM5RunRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM5RunRetention (void)

Function description

Get the SRAM5 retention in Run mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM5_RUN_NO_RETENTION
  – LL_PWR_SRAM5_RUN_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 SRAM5PD LL_PWR_GetSRAM5RunRetention
-

LL_PWR_SetSRAM6RunRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM6RunRetention (uint32_t SRAM6Retention)

Function description

Set the SRAM6 retention in Run mode.

Parameters

- **SRAM6Retention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM6_RUN_NO_RETENTION
  – LL_PWR_SRAM6_RUN_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SRAM6PD LL_PWR_SetSRAM6RunRetention
-

LL_PWR_GetSRAM6RunRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM6RunRetention (void)

Function description

Get the SRAM6 retention in Run mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM6_RUN_NO_RETENTION
  – LL_PWR_SRAM6_RUN_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR1 SRAM6PD LL_PWR_GetSRAM6RunRetention
-

LL_PWR_EnableOTGHSPHYLowPowerRetention

Function name

__STATIC_INLINE void LL_PWR_EnableOTGHSPHYLowPowerRetention (void)

Function description

Enable OTG_HS PHY power during low power modes (Stop2, Stop 3 and Standby).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FORCE_USBPWR LL_PWR_EnableOTGHSPHYLowPowerRetention

LL_PWR_DisableOTGHSPHYLowPowerRetention

Function name

__STATIC_INLINE void LL_PWR_DisableOTGHSPHYLowPowerRetention (void)

Function description

Disable OTG_HS PHY power during low power modes (Stop2, Stop 3 and Standby).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FORCE_USBPWR LL_PWR_DisableOTGHSPHYLowPowerRetention

LL_PWR_IsEnabledOTGHSPHYLowPowerRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledOTGHSPHYLowPowerRetention (void)

Function description

Check if OTG_HS PHY power during low power modes (Stop2, Stop 3 and Standby) is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FORCE_USBPWR LL_PWR_IsEnabledOTGHSPHYLowPowerRetention

LL_PWR_SetSRAM1StopRetention_1_3

Function name

__STATIC_INLINE void LL_PWR_SetSRAM1StopRetention_1_3 (uint32_t SRAM1PageRetention)

Function description

Set the SRAM1 page(s) (From page 1 to page 3) retention in Stop mode.

Parameters

- **SRAM1PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM1_STOP_NO_RETENTION
  – LL_PWR_SRAM1_STOP_1_3_RETENTION Or can be a combination of the following values:
  – LL_PWR_SRAM1_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE3_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM1PDS1 LL_PWR_SetSRAM1StopRetention_1_3
-
- CR2 SRAM1PDS2 LL_PWR_SetSRAM1StopRetention_1_3
-
- CR2 SRAM1PDS3 LL_PWR_SetSRAM1StopRetention_1_3

LL_PWR_SetSRAM1StopRetention_4_12

Function name

__STATIC_INLINE void LL_PWR_SetSRAM1StopRetention_4_12 (uint32_t SRAM1PageRetention)

Function description

Set the SRAM1 page(s) (From page 4 to page 12) retention in Stop mode.

Parameters

- **SRAM1PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM1_STOP_NO_RETENTION
  – LL_PWR_SRAM1_STOP_4_12_RETENTION Or can be a combination of the following values:
  – LL_PWR_SRAM1_STOP_PAGE4_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE5_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE6_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE7_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE8_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE9_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE10_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE11_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE12_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM1PDS4 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS5 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS6 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS7 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS8 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS9 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS10 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS11 LL_PWR_SetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS12 LL_PWR_SetSRAM1StopRetention_4_12

LL_PWR_GetSRAM1StopRetention_1_3

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM1StopRetention_1_3 (void)

Function description

Get the SRAM1 page(s) (From page 1 to page 3) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM1_STOP_NO_RETENTION
  – LL_PWR_SRAM1_STOP_1_3_RETENTION Or a combination of the following values:
  – LL_PWR_SRAM1_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM1_STOP_PAGE3_RETENTION

Reference Manual to LL API cross reference:

- CR2 SRAM1PDS1 LL_PWR_GetSRAM1StopRetention_1_3
-
- CR2 SRAM1PDS2 LL_PWR_GetSRAM1StopRetention_1_3
-
- CR2 SRAM1PDS3 LL_PWR_GetSRAM1StopRetention_1_3

LL_PWR_GetSRAM1StopRetention_4_12

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM1StopRetention_4_12 (void)

Function description

Get the SRAM1 page(s) (From page 4 to page 12) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM1_STOP_NO_RETENTION
  - LL_PWR_SRAM1_STOP_4_12_RETENTION Or a combination of the following values:
  - LL_PWR_SRAM1_STOP_PAGE4_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE5_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE6_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE7_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE8_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE9_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE10_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE11_RETENTION
  - LL_PWR_SRAM1_STOP_PAGE12_RETENTION

Reference Manual to LL API cross reference:

- CR2 SRAM1PDS4 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS5 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS6 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS7 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS8 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS9 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS10 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS11 LL_PWR_GetSRAM1StopRetention_4_12
-
- CR2 SRAM1PDS12 LL_PWR_GetSRAM1StopRetention_4_12

LL_PWR_SetSRAM2StopRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM2StopRetention (uint32_t SRAM2PageRetention)

Function description

Set the SRAM2 page(s) retention in Stop mode.

Parameters

- **SRAM2PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM2_STOP_NO_RETENTION
  – LL_PWR_SRAM2_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM2_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM2_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM2PDS1 LL_PWR_SetSRAM2StopRetention
-
- CR2 SRAM2PDS2 LL_PWR_SetSRAM2StopRetention

LL_PWR_GetSRAM2StopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM2StopRetention (void)

Function description

Get the SRAM2 page(s) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM2_STOP_NO_RETENTION
  – LL_PWR_SRAM2_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM2_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM2_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 SRAM2PDS1 LL_PWR_GetSRAM2StopRetention
-
- CR2 SRAM2PDS2 LL_PWR_GetSRAM2StopRetention

LL_PWR_SetSRAM3StopRetention_1_8

Function name

__STATIC_INLINE void LL_PWR_SetSRAM3StopRetention_1_8 (uint32_t SRAM3PageRetention)

Function description

Set the SRAM3 page(s) (From page 1 to page 8) retention in Stop mode.

Parameters

- **SRAM3PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM3_STOP_NO_RETENTION
  – LL_PWR_SRAM3_STOP_1_8_RETENTION Or can be a combination of the following values:
  – LL_PWR_SRAM3_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE3_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE4_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE5_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE6_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE7_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE8_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM3PDS1 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS2 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS3 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS4 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS5 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS6 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS7 LL_PWR_SetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS8 LL_PWR_SetSRAM3StopRetention_1_8

LL_PWR_SetSRAM3StopRetention_9_13

Function name

__STATIC_INLINE void LL_PWR_SetSRAM3StopRetention_9_13 (uint32_t SRAM3PageRetention)

Function description

Set the SRAM3 page(s) (From page 9 to page 13) retention in Stop mode.

Parameters

- **SRAM3PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_SRAM3_STOP_NO_RETENTION
  – LL_PWR_SRAM3_STOP_9_13_RETENTION Or can be a combination of the following values:
  – LL_PWR_SRAM3_STOP_PAGE9_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE10_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE11_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE12_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE13_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM3PDS9 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS10 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS11 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS12 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS13 LL_PWR_SetSRAM3StopRetention_9_13

LL_PWR_GetSRAM3StopRetention_1_8

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM3StopRetention_1_8 (void)

Function description

Get the SRAM3 page(s) (From page 1 to page 8) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM3_STOP_NO_RETENTION
  – LL_PWR_SRAM3_STOP_1_8_RETENTION Or can be a combination of the following values:
  – LL_PWR_SRAM3_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE3_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE4_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE5_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE6_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE7_RETENTION
  – LL_PWR_SRAM3_STOP_PAGE8_RETENTION

Reference Manual to LL API cross reference:

- CR2 SRAM3PDS1 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS2 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS3 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS4 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS5 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS6 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS7 LL_PWR_GetSRAM3StopRetention_1_8
-
- CR2 SRAM3PDS8 LL_PWR_GetSRAM3StopRetention_1_8

LL_PWR_GetSRAM3StopRetention_9_13

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM3StopRetention_9_13 (void)

Function description

Get the SRAM3 page(s) (From page 9 to page 13) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM3_STOP_NO_RETENTION
  - LL_PWR_SRAM3_STOP_9_13_RETENTION Or can be a combination of the following values:
  - LL_PWR_SRAM3_STOP_PAGE9_RETENTION
  - LL_PWR_SRAM3_STOP_PAGE10_RETENTION
  - LL_PWR_SRAM3_STOP_PAGE11_RETENTION
  - LL_PWR_SRAM3_STOP_PAGE12_RETENTION
  - LL_PWR_SRAM3_STOP_PAGE13_RETENTION

Reference Manual to LL API cross reference:

- CR2 SRAM3PDS9 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS10 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS11 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS12 LL_PWR_SetSRAM3StopRetention_9_13
-
- CR2 SRAM3PDS13 LL_PWR_SetSRAM3StopRetention_9_13

LL_PWR_SetSRAM4StopRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM4StopRetention (uint32_t SRAM4PageRetention)

Function description

Set the SRAM4 page retention in Stop mode.

Parameters

- **SRAM4PageRetention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM4_STOP_NO_RETENTION
  - LL_PWR_SRAM4_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM4PDS LL_PWR_SetSRAM4StopRetention
-

LL_PWR_GetSRAM4StopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM4StopRetention (void)

Function description

Get the SRAM4 page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM4_STOP_NO_RETENTION
  - LL_PWR_SRAM4_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 SRAM4PDS LL_PWR_GetSRAM4StopRetention

LL_PWR_SetSRAM5StopRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM5StopRetention (uint32_t SRAM5PageRetention)

Function description

Set the SRAM5 page(s) retention in Stop mode.

Parameters

- **SRAM5PageRetention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM5_STOP_NO_RETENTION
  - LL_PWR_SRAM5_STOP_FULL_RETENTION Or can be a combination of the following values:
  - LL_PWR_SRAM5_STOP_PAGE1_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE2_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE3_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE4_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE5_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE6_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE7_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE8_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE9_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE10_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE11_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE12_RETENTION
  - LL_PWR_SRAM5_STOP_PAGE13_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 SRAM5PDS1 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS2 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS3 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS4 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS5 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS6 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS7 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS8 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS9 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS10 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS11 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS12 LL_PWR_SetSRAM5StopRetention
-
- CR4 SRAM5PDS13 LL_PWR_SetSRAM5StopRetention

LL_PWR_GetSRAM5StopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM5StopRetention (void)

Function description

Get the SRAM5 page(s) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRAM5_STOP_NO_RETENTION
  – LL_PWR_SRAM5_STOP_FULL_RETENTION Or can be a combination of the following values:
  – LL_PWR_SRAM5_STOP_PAGE1_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE2_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE3_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE4_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE5_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE6_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE7_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE8_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE9_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE10_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE11_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE12_RETENTION
  – LL_PWR_SRAM5_STOP_PAGE13_RETENTION

Reference Manual to LL API cross reference:

- CR4 SRAM5PDS1 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS2 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS3 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS4 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS5 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS6 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS7 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS8 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS9 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS10 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS11 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS12 LL_PWR_GetSRAM5StopRetention
-
- CR4 SRAM5PDS13 LL_PWR_GetSRAM5StopRetention

LL_PWR_SetSRAM6StopRetention

Function name

__STATIC_INLINE void LL_PWR_SetSRAM6StopRetention (uint32_t SRAM6PageRetention)

Function description

Set the SRAM6 page(s) retention in Stop mode.

Parameters

- **SRAM6PageRetention:** : This parameter can be one of the following values:
  - LL_PWR_SRAM6_STOP_NO_RETENTION
  - LL_PWR_SRAM6_STOP_FULL_RETENTION Or can be a combination of the following values:
  - LL_PWR_SRAM6_STOP_PAGE1_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE2_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE3_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE4_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE5_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE6_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE7_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE8_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 SRAM6PDS1 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS2 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS3 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS4 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS5 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS6 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS7 LL_PWR_SetSRAM6StopRetention
-
- CR5 SRAM6PDS8 LL_PWR_SetSRAM6StopRetention

LL_PWR_GetSRAM6StopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSRAM6StopRetention (void)

Function description

Get the SRAM6 page(s) retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_SRAM6_STOP_NO_RETENTION
  - LL_PWR_SRAM6_STOP_FULL_RETENTION Or can be a combination of the following values:
  - LL_PWR_SRAM6_STOP_PAGE1_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE2_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE3_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE4_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE5_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE6_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE7_RETENTION
  - LL_PWR_SRAM6_STOP_PAGE8_RETENTION

Reference Manual to LL API cross reference:

- CR5 SRAM6PDS1 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS2 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS3 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS4 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS5 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS6 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS7 LL_PWR_GetSRAM6StopRetention
-
- CR5 SRAM6PDS8 LL_PWR_GetSRAM6StopRetention

LL_PWR_SetICacheRAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetICacheRAMStopRetention (uint32_t ICachePageRetention)

Function description

Set the ICACHE SRAM page retention in Stop mode.

Parameters

- **ICachePageRetention:** : This parameter can be one of the following values:
  - LL_PWR_ICACHERAM_STOP_NO_RETENTION
  - LL_PWR_ICACHERAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 ICRAMPDS LL_PWR_SetICacheRAMStopRetention
-

LL_PWR_GetICacheRAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetICacheRAMStopRetention (void)

Function description

Get the ICACHE SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_ICACHERAM_STOP_NO_RETENTION
  – LL_PWR_ICACHERAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 ICRAMPDS LL_PWR_GetICacheRAMStopRetention

LL_PWR_SetDCache1RAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetDCache1RAMStopRetention (uint32_t DCache1PageRetention)

Function description

Set the DCACHE1 SRAM page retention in Stop mode.

Parameters

- **DCache1PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_DCACHE1RAM_STOP_NO_RETENTION
  – LL_PWR_DCACHE1RAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DC1RAMPDS LL_PWR_SetDCache1RAMStopRetention
-

LL_PWR_GetDCache1RAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetDCache1RAMStopRetention (void)

Function description

Get the DCACHE1 SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_DCACHE1RAM_STOP_NO_RETENTION
  – LL_PWR_DCACHE1RAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 DC1RAMPDS LL_PWR_GetDCache1RAMStopRetention

LL_PWR_SetDCache2RAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetDCache2RAMStopRetention (uint32_t DCache2PageRetention)

Function description

Set the DCACHE2 SRAM page retention in Stop mode.

Parameters

- **DCache2PageRetention:** : This parameter can be one of the following values:
  – LL_PWR_DCACHE2RAM_STOP_NO_RETENTION
  – LL_PWR_DCACHE2RAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DC2RAMPDS LL_PWR_SetDCache2RAMStopRetention
-

LL_PWR_GetDCache2RAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetDCache2RAMStopRetention (void)

Function description

Get the DCACHE2 SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_DCACHE2RAM_STOP_NO_RETENTION
  – LL_PWR_DCACHE2RAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 DC2RAMPDS LL_PWR_GetDCache2RAMStopRetention

LL_PWR_SetDMA2DRAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetDMA2DRAMStopRetention (uint32_t DMA2DRAMPageRetention)

Function description

Set the DMA2D SRAM page retention in Stop mode.

Parameters

- **DMA2DRAMPageRetention:** : This parameter can be one of the following values:
  – LL_PWR_DMA2DRAM_STOP_NO_RETENTION
  – LL_PWR_DMA2DRAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DMA2DRAMPDS LL_PWR_SetDMA2DRAMStopRetention
-

LL_PWR_GetDMA2DRAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetDMA2DRAMStopRetention (void)

Function description

Get the DMA2D SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_DMA2DRAM_STOP_NO_RETENTION
  - LL_PWR_DMA2DRAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 DMA2DRAMPDS LL_PWR_GetDMA2DRAMStopRetention

LL_PWR_SetPeriphRAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetPeriphRAMStopRetention (uint32_t PriphRAMPageRetention)

Function description

Set the FMAC, FDCAN and USB SRAMs pages retention in Stop mode.

Parameters

- **PriphRAMPageRetention:** : This parameter can be one of the following values:
  - LL_PWR_PERIPHRAM_STOP_NO_RETENTION
  - LL_PWR_PERIPHRAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PRAMPDS LL_PWR_SetPeriphRAMStopRetention
-

LL_PWR_GetPeriphRAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPeriphRAMStopRetention (void)

Function description

Get the FMAC, FDCAN and USB SRAMs pages retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PERIPHRAM_STOP_NO_RETENTION
  - LL_PWR_PERIPHRAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 PRAMPDS LL_PWR_GetPeriphRAMStopRetention

LL_PWR_SetPKARAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetPKARAMStopRetention (uint32_t PKARAMPageRetention)

Function description

Set the PKA32 SRAM page retention in Stop mode.

Parameters

- **PKARAMPageRetention:** : This parameter can be one of the following values:
  - LL_PWR_PKARAM_STOP_NO_RETENTION
  - LL_PWR_PKARAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PKARAMPDS LL_PWR_SetPKARAMStopRetention

LL_PWR_GetPKARAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPKARAMStopRetention (void)

Function description

Get the PKA32 SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PKARAM_STOP_NO_RETENTION
  - LL_PWR_PKARAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 PKARAMPDS LL_PWR_GetPKARAMStopRetention

LL_PWR_SetGraphicPeriphRAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetGraphicPeriphRAMStopRetention (uint32_t GraphicPeriphRAMPageRetention)

Function description

Set the Graphic peripherals (LTDC, GFXMMU) SRAM page retention in Stop mode.

Parameters

- **GraphicPeriphRAMPageRetention:** : This parameter can be one of the following values:
  - LL_PWR_GRAPHICPERIPHRAM_STOP_NO_RETENTION
  - LL_PWR_GRAPHICPERIPHRAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 GPRAMPDS LL_PWR_SetGraphicPeriphRAMStopRetention

LL_PWR_GetGraphicPeriphRAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetGraphicPeriphRAMStopRetention (void)

Function description

Get the Graphic peripherals (LTDC, GFXMMU) SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_GRAPHICPERIPHRAM_STOP_NO_RETENTION
  – LL_PWR_GRAPHICPERIPHRAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 GPRAMPDS LL_PWR_GetGraphicPeriphRAMStopRetention

LL_PWR_SetDSIRAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetDSIRAMStopRetention (uint32_t DSIRAMPageRetention)

Function description

Set the DSI SRAM page retention in Stop mode.

Parameters

- **DSIRAMPageRetention:** : This parameter can be one of the following values:
  – LL_PWR_DSIRAM_STOP_NO_RETENTION
  – LL_PWR_DSIRAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DSIRAMPDS LL_PWR_SetDSIRAMStopRetention

LL_PWR_GetDSIRAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetDSIRAMStopRetention (void)

Function description

Get the DSI SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_DSIRAM_STOP_NO_RETENTION
  – LL_PWR_DSIRAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 DSIRAMPDS LL_PWR_GetDSIRAMStopRetention

LL_PWR_SetJPEGRAMStopRetention

Function name

__STATIC_INLINE void LL_PWR_SetJPEGRAMStopRetention (uint32_t JPEGRAMPageRetention)

Function description

Set the JPEG SRAM page retention in Stop mode.

Parameters

- **JPEGRAMPageRetention:** : This parameter can be one of the following values:
  – LL_PWR_JPEGRAM_STOP_NO_RETENTION
  – LL_PWR_JPEGRAM_STOP_FULL_RETENTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 JPEGRAMPDS LL_PWR_SetJPEGRAMStopRetention

LL_PWR_GetJPEGRAMStopRetention

Function name

__STATIC_INLINE uint32_t LL_PWR_GetJPEGRAMStopRetention (void)

Function description

Get the JPEG SRAM page retention in Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_JPEGRAM_STOP_NO_RETENTION
  – LL_PWR_JPEGRAM_STOP_FULL_RETENTION

Reference Manual to LL API cross reference:

- CR2 JPEGRAMPDS LL_PWR_GetJPEGRAMStopRetention

LL_PWR_EnableFlashFastWakeUp

Function name

__STATIC_INLINE void LL_PWR_EnableFlashFastWakeUp (void)

Function description

Enable the flash memory fast wakeup from Stop mode (Stop 0, 1).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 FLASHFWU LL_PWR_EnableFlashFastWakeUp

LL_PWR_DisableFlashFastWakeUp

Function name

__STATIC_INLINE void LL_PWR_DisableFlashFastWakeUp (void)

Function description

Disable the flash memory fast wakeup from Stop mode (Stop 0, 1).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 FLASHFWU LL_PWR_DisableFlashFastWakeUp

LL_PWR_IsEnabledFlashFastWakeUp

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledFlashFastWakeUp (void)

Function description

Check if the flash memory fast wakeup from Stop mode (Stop 0, 1) is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 FLASHFWU LL_PWR_IsEnabledFlashFastWakeUp

LL_PWR_EnableSRAM4FastWakeUp

Function name

__STATIC_INLINE void LL_PWR_EnableSRAM4FastWakeUp (void)

Function description

Enable the SRAM4 memory fast wakeup from Stop mode (Stop 0, 1, 2).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM4FWU LL_PWR_EnableSRAM4FastWakeUp

LL_PWR_DisableSRAM4FastWakeUp

Function name

__STATIC_INLINE void LL_PWR_DisableSRAM4FastWakeUp (void)

Function description

Disable the SRAM4 memory fast wakeup from Stop mode (Stop 0, 1, 2).

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRAM4FWU LL_PWR_DisableSRAM4FastWakeUp

LL_PWR_IsEnabledSRAM4FastWakeUp

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledSRAM4FastWakeUp (void)

Function description

Check if the SRAM4 memory fast wakeup from Stop mode (Stop 0, 1, 2).

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 SRAM4FWU LL_PWR_IsEnabledSRAM4FastWakeUp

LL_PWR_SetSmartRunDomainMode

Function name

__STATIC_INLINE void LL_PWR_SetSmartRunDomainMode (uint32_t SRDMode)

Function description

Set the smart run domain mode.

Parameters

- **SRDMode:** : This parameter can be one of the following values:
  – LL_PWR_SRD_RUN_MODE
  – LL_PWR_SRD_STOP_MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SRDRUN LL_PWR_SetSmartRunDomainMode

LL_PWR_GetSmartRunDomainMode

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSmartRunDomainMode (void)

Function description

Get the smart run domain mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_SRD_RUN_MODE
  – LL_PWR_SRD_STOP_MODE

Reference Manual to LL API cross reference:

- CR2 SRDRUN LL_PWR_GetSmartRunDomainMode

LL_PWR_SetRegulatorSupply

Function name

__STATIC_INLINE void LL_PWR_SetRegulatorSupply (uint32_t RegulatorSupply)

Function description

Set the VCore regulator supply.

Parameters

- **RegulatorSupply:** : This parameter can be one of the following values:
  – LL_PWR_LDO_SUPPLY
  – LL_PWR_SMPS_SUPPLY

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 REGSEL LL_PWR_SetRegulatorSupply

LL_PWR_GetRegulatorSupply

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulatorSupply (void)

Function description

Get the VCore regulator supply.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_LDO_SUPPLY
  – LL_PWR_SMPS_SUPPLY

Reference Manual to LL API cross reference:

- CR3 REGSEL LL_PWR_GetRegulatorSupply

LL_PWR_EnableFastSoftStart

Function name

__STATIC_INLINE void LL_PWR_EnableFastSoftStart (void)

Function description

Enable the fast soft start for selected regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 FSTEN LL_PWR_EnableFastSoftStart

LL_PWR_DisableFastSoftStart

Function name

__STATIC_INLINE void LL_PWR_DisableFastSoftStart (void)

Function description

Disable the fast soft start for selected regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 FSTEN LL_PWR_DisableFastSoftStart

LL_PWR_IsEnabledFastSoftStart

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledFastSoftStart (void)

Function description

Check if the fast soft start for selected regulator is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 FSTEN LL_PWR_IsEnabledFastSoftStart

LL_PWR_SetRegulVoltageScaling

Function name

__STATIC_INLINE void LL_PWR_SetRegulVoltageScaling (uint32_t VoltageScaling)

Function description

Set the regulator supply output voltage.

Parameters

- **VoltageScaling:** This parameter can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3
  - LL_PWR_REGU_VOLTAGE_SCALE4

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR VOS LL_PWR_SetRegulVoltageScaling

LL_PWR_GetRegulVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulVoltageScaling (void)

Function description

Get the regulator supply output voltage.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_REGU_VOLTAGE_SCALE1
  - LL_PWR_REGU_VOLTAGE_SCALE2
  - LL_PWR_REGU_VOLTAGE_SCALE3
  - LL_PWR_REGU_VOLTAGE_SCALE4

Reference Manual to LL API cross reference:

- VOSR VOS LL_PWR_GetRegulVoltageScaling

LL_PWR_EnableEPODBooster

Function name

__STATIC_INLINE void LL_PWR_EnableEPODBooster (void)

Function description

Enable the EPOD (Embedded Power Distribution) booster.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR BOOSTEN LL_PWR_EnableEPODBooster

LL_PWR_DisableEPODBooster

Function name

__STATIC_INLINE void LL_PWR_DisableEPODBooster (void)

Function description

Disable the EPOD (Embedded Power Distribution) booster.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR BOOSTEN LL_PWR_DisableEPODBooster

LL_PWR_IsEnabledEPODBooster

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledEPODBooster (void)

Function description

Check if the EPOD (Embedded Power Distribution) booster is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR BOOSTEN LL_PWR_IsEnabledEPODBooster

LL_PWR_EnableUSBPowerSupply

Function name

__STATIC_INLINE void LL_PWR_EnableUSBPowerSupply (void)

Function description

Enable the USB power supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR USBPWREN LL_PWR_EnableUSBPowerSupply

LL_PWR_DisableUSBPowerSupply

Function name

__STATIC_INLINE void LL_PWR_DisableUSBPowerSupply (void)

Function description

Disable the USB power supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR USBPWREN LL_PWR_DisableUSBPowerSupply

LL_PWR_IsEnabledUSBPowerSupply

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUSBPowerSupply (void)

Function description

Check if the USB power supply is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR USBPWREN LL_PWR_IsEnabledUSBPowerSupply

LL_PWR_EnableUSBEPODBooster

Function name

__STATIC_INLINE void LL_PWR_EnableUSBEPODBooster (void)

Function description

Enable the USB EPOD (Embedded Power Distribution) booster.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR USBBOOSTEN LL_PWR_EnableUSBEPODBooster

LL_PWR_DisableUSBEPODBooster

Function name

__STATIC_INLINE void LL_PWR_DisableUSBEPODBooster (void)

Function description

Disable the USB EPOD (Embedded Power Distribution) booster.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR USBBOOSTEN LL_PWR_DisableUSBEPODBooster

LL_PWR_IsEnabledUSBEPODBooster

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUSBEPODBooster (void)

Function description

Check if the USB EPOD (Embedded Power Distribution) booster is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR USBBOOSTEN LL_PWR_IsEnabledUSBEPODBooster

LL_PWR_EnableVDD11USB

Function name

__STATIC_INLINE void LL_PWR_EnableVDD11USB (void)

Function description

Enable the VDD11USB.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR VDD11USBDIS LL_PWR_EnableVDD11USB

LL_PWR_DisableVDD11USB

Function name

__STATIC_INLINE void LL_PWR_DisableVDD11USB (void)

Function description

Disable the VDD11USB.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VOSR VDD11USBDIS LL_PWR_DisableVDD11USB

LL_PWR_IsEnabledVDD11USB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVDD11USB (void)

Function description

Check if the VDD11USB is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR VDD11USBDIS LL_PWR_IsEnabledVDD11USB

LL_PWR_SetPVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)

Function description

Set the Power voltage detector level.

Parameters

- **PVDLevel:** This parameter can be one of the following values:
  – LL_PWR_PVDLEVEL_0
  – LL_PWR_PVDLEVEL_1
  – LL_PWR_PVDLEVEL_2
  – LL_PWR_PVDLEVEL_3
  – LL_PWR_PVDLEVEL_4
  – LL_PWR_PVDLEVEL_5
  – LL_PWR_PVDLEVEL_6
  – LL_PWR_PVDLEVEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR PVDLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)

Function description

Get the Power voltage detector level.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:

- SVMCR PVDLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name

__STATIC_INLINE void LL_PWR_EnablePVD (void)

Function description

Enable the power voltage detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR PVDE LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name

__STATIC_INLINE void LL_PWR_DisablePVD (void)

Function description

Disable the power voltage detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR PVDE LL_PWR_DisablePVD

LL_PWR_IsEnabledPVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)

Function description

Check if the power voltage detector is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR PVDE LL_PWR_IsEnabledPVD

LL_PWR_EnableVddUSB

Function name

__STATIC_INLINE void LL_PWR_EnableVddUSB (void)

Function description

Enable the independent USB supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR USV LL_PWR_EnableVddUSB

LL_PWR_DisableVddUSB

Function name

__STATIC_INLINE void LL_PWR_DisableVddUSB (void)

Function description

Disable the independent USB supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR USV LL_PWR_DisableVDDUSB

LL_PWR_IsEnabledVddUSB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddUSB (void)

Function description

Check if the independent USB supply is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR USV LL_PWR_IsEnabledVddUSB

LL_PWR_EnableVddIO2

Function name

__STATIC_INLINE void LL_PWR_EnableVddIO2 (void)

Function description

Enable the independent I/Os supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR IO2SV LL_PWR_EnableVddIO2

LL_PWR_DisableVddIO2

Function name

__STATIC_INLINE void LL_PWR_DisableVddIO2 (void)

Function description

Disable the independent I/Os supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR IO2SV LL_PWR_DisableVddIO2

LL_PWR_IsEnabledVddIO2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddIO2 (void)

Function description

Check if the independent I/Os supply is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR IO2SV LL_PWR_IsEnabledVddIO2

LL_PWR_EnableVddA

Function name

__STATIC_INLINE void LL_PWR_EnableVddA (void)

Function description

Enable the independent analog supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR ASV LL_PWR_EnableVddA

LL_PWR_DisableVddA

Function name

__STATIC_INLINE void LL_PWR_DisableVddA (void)

Function description

Disable the independent analog supply.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR ASV LL_PWR_DisableVddA

LL_PWR_IsEnabledVddA

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddA (void)

Function description

Check if the independent analog supply is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR ASV LL_PWR_IsEnabledVddA

LL_PWR_EnableVddUSBMonitor

Function name

__STATIC_INLINE void LL_PWR_EnableVddUSBMonitor (void)

Function description

Enable the independent USB supply monitor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR UVMEN LL_PWR_EnableVddUSBMonitor

LL_PWR_DisableVddUSBMonitor

Function name

__STATIC_INLINE void LL_PWR_DisableVddUSBMonitor (void)

Function description

Disable the independent USB supply monitor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR UVMEN LL_PWR_DisableVddUSBMonitor

LL_PWR_IsEnabledVddUSBMonitor

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddUSBMonitor (void)

Function description

Check if the independent USB supply monitor is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR UVMEN LL_PWR_IsEnabledVddUSBMonitor

LL_PWR_EnableVddIO2Monitor

Function name

__STATIC_INLINE void LL_PWR_EnableVddIO2Monitor (void)

Function description

Enable the independent I/Os supply monitor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR IO2VMEN LL_PWR_EnableVddIO2Monitor

LL_PWR_DisableVddIO2Monitor

Function name

__STATIC_INLINE void LL_PWR_DisableVddIO2Monitor (void)

Function description

Disable the independent I/Os supply monitor.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR IO2VMEN LL_PWR_DisableVddIO2Monitor

LL_PWR_IsEnabledVddIO2Monitor

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddIO2Monitor (void)

Function description

Check if the independent I/Os supply monitor is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR IO2VMEN LL_PWR_IsEnabledVddIO2Monitor

LL_PWR_EnableVddAMonitor1

Function name

__STATIC_INLINE void LL_PWR_EnableVddAMonitor1 (void)

Function description

Enable the independent analog supply monitor 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR AVM1EN LL_PWR_EnableVddAMonitor1

LL_PWR_DisableVddAMonitor1

Function name

__STATIC_INLINE void LL_PWR_DisableVddAMonitor1 (void)

Function description

Disable the independent analog supply monitor 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR AVM1EN LL_PWR_DisableVddAMonitor1

LL_PWR_IsEnabledVddAMonitor1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddAMonitor1 (void)

Function description

Check if the independent analog supply monitor 1 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR AVM1EN LL_PWR_IsEnabledVddAMonitor1

LL_PWR_EnableVddAMonitor2

Function name

__STATIC_INLINE void LL_PWR_EnableVddAMonitor2 (void)

Function description

Enable the independent analog supply monitor 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR AVM2EN LL_PWR_EnableVddAMonitor2

LL_PWR_DisableVddAMonitor2

Function name

__STATIC_INLINE void LL_PWR_DisableVddAMonitor2 (void)

Function description

Disable the independent analog supply monitor 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SVMCR AVM2EN LL_PWR_DisableVddAMonitor2

LL_PWR_IsEnabledVddAMonitor2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledVddAMonitor2 (void)

Function description

Check if the independent analog supply monitor 2 is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMCR AVM2EN LL_PWR_IsEnabledVddAMonitor2

LL_PWR_EnableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable the wake up pin_x.

Parameters

- **WakeUpPin:** This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR1 WUPENx LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the wake up pin_x.

Parameters

- **WakeUpPin:** This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR1 WUPENx LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)

Function description

Check if the wake up pin_x is enabled.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUCR1 WUPENx LL_PWR_IsEnabledWakeUpPin

LL_PWR_SetWakeUpPinPolarityLow

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Set the wake up pin polarity low for the event detection.

Parameters

- **WakeUpPin:** This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR2 WUPPx LL_PWR_SetWakeUpPinPolarityLow

LL_PWR_SetWakeUpPinPolarityHigh

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinPolarityHigh (uint32_t WakeUpPin)

Function description

Set the wake up pin polarity high for the event detection.

Parameters

- **WakeUpPin:** This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR2 WUPPx LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_GetWakeUpPinPolarity

Function name

__STATIC_INLINE uint32_t LL_PWR_GetWakeUpPinPolarity (uint32_t WakeUpPin)

Function description

Get the wake up pin polarity for the event detection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  – LL_PWR_WAKEUP_PIN1
  – LL_PWR_WAKEUP_PIN2
  – LL_PWR_WAKEUP_PIN3
  – LL_PWR_WAKEUP_PIN4
  – LL_PWR_WAKEUP_PIN5
  – LL_PWR_WAKEUP_PIN6
  – LL_PWR_WAKEUP_PIN7
  – LL_PWR_WAKEUP_PIN8

Return values

- **State:** of bit (1 : polarity or 0 : polarity high).

Reference Manual to LL API cross reference:

- WUCR2 WUPPx LL_PWR_GetWakeUpPinPolarity

LL_PWR_SetWakeUpPinSignal0Selection

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinSignal0Selection (uint32_t WakeUpPin)

Function description

Set the wakeup pin_x selection 0.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  – LL_PWR_WAKEUP_PIN1
  – LL_PWR_WAKEUP_PIN2
  – LL_PWR_WAKEUP_PIN3
  – LL_PWR_WAKEUP_PIN4
  – LL_PWR_WAKEUP_PIN5
  – LL_PWR_WAKEUP_PIN6
  – LL_PWR_WAKEUP_PIN7
  – LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR3 WUSELx LL_PWR_SetWakeUpPinSignal0Selection

LL_PWR_SetWakeUpPinSignal1Selection

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinSignal1Selection (uint32_t WakeUpPin)

Function description

Set the wakeup pin_x selection 1.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  – LL_PWR_WAKEUP_PIN1
  – LL_PWR_WAKEUP_PIN2
  – LL_PWR_WAKEUP_PIN3
  – LL_PWR_WAKEUP_PIN4
  – LL_PWR_WAKEUP_PIN5
  – LL_PWR_WAKEUP_PIN6
  – LL_PWR_WAKEUP_PIN7
  – LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR3 WUSELx LL_PWR_SetWakeUpPinSignal1Selection

LL_PWR_SetWakeUpPinSignal2Selection

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinSignal2Selection (uint32_t WakeUpPin)

Function description

Set the wakeup pin_x selection 2.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  – LL_PWR_WAKEUP_PIN1
  – LL_PWR_WAKEUP_PIN2
  – LL_PWR_WAKEUP_PIN3
  – LL_PWR_WAKEUP_PIN4
  – LL_PWR_WAKEUP_PIN5
  – LL_PWR_WAKEUP_PIN6
  – LL_PWR_WAKEUP_PIN7
  – LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR3 WUSELx LL_PWR_SetWakeUpPinSignal2Selection

LL_PWR_SetWakeUpPinSignal3Selection

Function name

__STATIC_INLINE void LL_PWR_SetWakeUpPinSignal3Selection (uint32_t WakeUpPin)

Function description

Set the wakeup pin_x selection 3.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUCR3 WUSELx LL_PWR_SetWakeUpPinSignal3Selection

LL_PWR_GetWakeUpPinSignalSelection

Function name

__STATIC_INLINE uint32_t LL_PWR_GetWakeUpPinSignalSelection (uint32_t WakeUpPin)

Function description

Get the wakeup pin_x selection.

Parameters

- **WakeUpPin:** This parameter can be one of the following values:
  - LL_PWR_WAKEUP_PIN1
  - LL_PWR_WAKEUP_PIN2
  - LL_PWR_WAKEUP_PIN3
  - LL_PWR_WAKEUP_PIN4
  - LL_PWR_WAKEUP_PIN5
  - LL_PWR_WAKEUP_PIN6
  - LL_PWR_WAKEUP_PIN7
  - LL_PWR_WAKEUP_PIN8

Reference Manual to LL API cross reference:

- WUCR3 WUSELx LL_PWR_GetWakeUpPinSignalSelection

LL_PWR_EnableBkUpRegulator

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpRegulator (void)

Function description

Enable backup regulator.

Return values

- **None:**

Notes

- When this bit is set, the backup RAM content is kept in Standby and VBAT modes. If BREN is reset, the backup RAM can still be used in Run, Sleep and Stop modes. However, its content is lost in Standby, Shutdown and VBAT modes. This bit can be written only when the regulator is LDO, which must be configured before switching to SMPS.

Reference Manual to LL API cross reference:

- BDCR1 BREN LL_PWR_EnableBkUpRegulator

LL_PWR_DisableBkUpRegulator

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpRegulator (void)

Function description

Disable backup regulator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR1 BREN LL_PWR_DisableBkUpRegulator

LL_PWR_IsEnabledBkUpRegulator

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpRegulator (void)

Function description

Check if the backup regulator is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR1 BREN LL_PWR_IsEnabledBkUpRegulator

LL_PWR_EnableMonitoring

Function name

__STATIC_INLINE void LL_PWR_EnableMonitoring (void)

Function description

Enable the backup domain voltage and temperature monitoring.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR1 MONEN LL_PWR_EnableMonitoring

LL_PWR_DisableMonitoring

Function name

__STATIC_INLINE void LL_PWR_DisableMonitoring (void)

Function description

Disable the backup domain voltage and temperature monitoring.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR1 MONEN LL_PWR_DisableMonitoring

LL_PWR_IsEnabledMonitoring

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledMonitoring (void)

Function description

Check if the backup domain voltage and temperature monitoring is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR1 MONEN LL_PWR_IsEnabledMonitoring

LL_PWR_EnableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_EnableBatteryCharging (void)

Function description

Enable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR2 VBE LL_PWR_EnableBatteryCharging

LL_PWR_DisableBatteryCharging

Function name

__STATIC_INLINE void LL_PWR_DisableBatteryCharging (void)

Function description

Disable battery charging.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR2 VBE LL_PWR_DisableBatteryCharging

LL_PWR_IsEnabledBatteryCharging

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBatteryCharging (void)

Function description

Check if battery charging is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR2 VBE LL_PWR_IsEnabledBatteryCharging

LL_PWR_SetBattChargResistor

Function name

__STATIC_INLINE void LL_PWR_SetBattChargResistor (uint32_t Resistor)

Function description

Set the Battery charge resistor impedance.

Parameters

- **Resistor:** This parameter can be one of the following values:
  – LL_PWR_BATT_CHARG_RESISTOR_5K
  – LL_PWR_BATT_CHARG_RESISTOR_1_5K

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR2 VBRS LL_PWR_SetBattChargResistor

LL_PWR_GetBattChargResistor

Function name

__STATIC_INLINE uint32_t LL_PWR_GetBattChargResistor (void)

Function description

Get the Battery charge resistor impedance.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_BATT_CHARG_RESISTOR_5K
  – LL_PWR_BATT_CHARG_RESISTOR_1_5K

Reference Manual to LL API cross reference:

- BDCR2 VBRS LL_PWR_GetBattChargResistor

LL_PWR_EnableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_EnableBkUpAccess (void)

Function description

Enable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBPR DBP LL_PWR_EnableBkUpAccess

LL_PWR_DisableBkUpAccess

Function name

__STATIC_INLINE void LL_PWR_DisableBkUpAccess (void)

Function description

Disable access to the backup domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBPR DBP LL_PWR_DisableBkUpAccess

LL_PWR_IsEnabledBkUpAccess

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBkUpAccess (void)

Function description

Check if the access to backup domain is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DBPR DBP LL_PWR_IsEnabledBkUpAccess

LL_PWR_EnableUCPDStandbyMode

Function name

__STATIC_INLINE void LL_PWR_EnableUCPDStandbyMode (void)

Function description

Enable the USB Type-C and Power Delivery memorization in Standby mode.

Return values

- **None:**

Notes

- This function must be called just before entering Standby mode.

Reference Manual to LL API cross reference:

- UCPDR UCPD_STDBY LL_PWR_EnableUCPDStandbyMode

LL_PWR_DisableUCPDStandbyMode

Function name

__STATIC_INLINE void LL_PWR_DisableUCPDStandbyMode (void)

Function description

Disable the USB Type-C and Power Delivery memorization in Standby mode.

Return values

- **None:**

Notes

- This function must be called after exiting Standby mode and before any UCPD configuration update.

Reference Manual to LL API cross reference:

- UCPDR UCPD_STDBY LL_PWR_DisableUCPDStandbyMode

LL_PWR_IsEnabledUCPDStandbyMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUCPDStandbyMode (void)

Function description

Check the USB Type-C and Power Delivery Standby mode memorization state.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- UCPDR UCPD_STDBY LL_PWR_IsEnabledUCPDStandbyMode

LL_PWR_EnableUCPDDeadBattery

Function name

__STATIC_INLINE void LL_PWR_EnableUCPDDeadBattery (void)

Function description

Enable the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- UCPDR UCPD_DBDIS LL_PWR_EnableUCPDDeadBattery

LL_PWR_DisableUCPDDeadBattery

Function name

__STATIC_INLINE void LL_PWR_DisableUCPDDeadBattery (void)

Function description

Disable the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **None:**

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- UCPDR UCPD_DBDIS LL_PWR_DisableUCPDDeadBattery

LL_PWR_IsEnabledUCPDDeadBattery

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledUCPDDeadBattery (void)

Function description

Check the USB Type-C and power delivery dead battery pull-down behavior on UCPD CC1 and CC2 pins.

Return values

- **State:** of feature (1 : enabled; 0 : disabled).

Notes

- After exiting reset, the USB Type-C dead battery behavior is enabled, which may have a pull-down effect on CC1 and CC2 pins. It is recommended to disable it in all cases, either to stop this pull-down or to hand over control to the UCPD (which should therefore be initialized before doing the disable).

Reference Manual to LL API cross reference:

- UCPDR UCPD_DBDIS LL_PWR_IsEnabledUCPDDeadBattery

LL_PWR_EnablePUPDConfig

Function name

__STATIC_INLINE void LL_PWR_EnablePUPDConfig (void)

Function description

Enable the pull-up and pull-down configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APCR APC LL_PWR_EnablePUPDConfig

LL_PWR_DisablePUPDConfig

Function name

__STATIC_INLINE void LL_PWR_DisablePUPDConfig (void)

Function description

Disable the pull-up and pull-down configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- APCR APC LL_PWR_DisablePUPDConfig

LL_PWR_IsEnabledPUPDConfig

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPUPDConfig (void)

Function description

Check if the pull-up and pull-down configuration is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- APCR APC LL_PWR_IsEnabledPUPDConfig

LL_PWR_EnableGPIOPullUp

Function name

__STATIC_INLINE void LL_PWR_EnableGPIOPullUp (uint32_t GPIOPort, uint32_t GPIOPin)

Function description

Enable GPIO pull-up in Standby and Shutdown modes.

Parameters

- **GPIOPort:** This parameter can be one of the following values:
  - LL_PWR_GPIO_PORTA
  - LL_PWR_GPIO_PORTB
  - LL_PWR_GPIO_PORTC
  - LL_PWR_GPIO_PORTD
  - LL_PWR_GPIO_PORTE
  - LL_PWR_GPIO_PORTF
  - LL_PWR_GPIO_PORTG
  - LL_PWR_GPIO_PORTH
  - LL_PWR_GPIO_PORTI
  - LL_PWR_GPIO_PORTJ (*) (*) value can be used only for STM32U59xxx and STM32U5A9xx devices.
- **GPIOPin:** This parameter can be a combination of the following values:
  - LL_PWR_GPIO_PIN_0
  - LL_PWR_GPIO_PIN_1
  - LL_PWR_GPIO_PIN_2
  - LL_PWR_GPIO_PIN_3
  - LL_PWR_GPIO_PIN_4
  - LL_PWR_GPIO_PIN_5
  - LL_PWR_GPIO_PIN_6
  - LL_PWR_GPIO_PIN_7
  - LL_PWR_GPIO_PIN_8
  - LL_PWR_GPIO_PIN_9
  - LL_PWR_GPIO_PIN_10
  - LL_PWR_GPIO_PIN_11
  - LL_PWR_GPIO_PIN_12
  - LL_PWR_GPIO_PIN_13
  - LL_PWR_GPIO_PIN_14
  - LL_PWR_GPIO_PIN_15

Return values

- **None.:**

Reference Manual to LL API cross reference:

- PUCRx PUx LL_PWR_EnableGPIOPullUp

LL_PWR_DisableGPIOPullUp

Function name

__STATIC_INLINE void LL_PWR_DisableGPIOPullUp (uint32_t GPIOPort, uint32_t GPIOPin)

Function description

Disable GPIO pull-up in Standby and Shutdown modes.

Parameters

- **GPIOPort:** This parameter can be one of the following values:
  – LL_PWR_GPIO_PORTA
  – LL_PWR_GPIO_PORTB
  – LL_PWR_GPIO_PORTC
  – LL_PWR_GPIO_PORTD
  – LL_PWR_GPIO_PORTE
  – LL_PWR_GPIO_PORTF
  – LL_PWR_GPIO_PORTG
  – LL_PWR_GPIO_PORTH
  – LL_PWR_GPIO_PORTI
  – LL_PWR_GPIO_PORTJ (*) (*) value can be used only for STM32U59xxx and STM32U5A9xx devices.
- **GPIOPin:** This parameter can be a combination of the following values:
  – LL_PWR_GPIO_PIN_0
  – LL_PWR_GPIO_PIN_1
  – LL_PWR_GPIO_PIN_2
  – LL_PWR_GPIO_PIN_3
  – LL_PWR_GPIO_PIN_4
  – LL_PWR_GPIO_PIN_5
  – LL_PWR_GPIO_PIN_6
  – LL_PWR_GPIO_PIN_7
  – LL_PWR_GPIO_PIN_8
  – LL_PWR_GPIO_PIN_9
  – LL_PWR_GPIO_PIN_10
  – LL_PWR_GPIO_PIN_11
  – LL_PWR_GPIO_PIN_12
  – LL_PWR_GPIO_PIN_13
  – LL_PWR_GPIO_PIN_14
  – LL_PWR_GPIO_PIN_15

Return values

- **None.:**

Reference Manual to LL API cross reference:

- PUCRx PUx LL_PWR_DisableGPIOPullUp

LL_PWR_IsEnabledGPIOPullUp

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullUp (uint32_t GPIOPort, uint32_t GPIOPin)

Function description

Check if GPIO pull-up in Standby and Shutdown modes is enabled.

Parameters

- **GPIOPort:** This parameter can be one of the following values:
  - LL_PWR_GPIO_PORTA
  - LL_PWR_GPIO_PORTB
  - LL_PWR_GPIO_PORTC
  - LL_PWR_GPIO_PORTD
  - LL_PWR_GPIO_PORTE
  - LL_PWR_GPIO_PORTF
  - LL_PWR_GPIO_PORTG
  - LL_PWR_GPIO_PORTH
  - LL_PWR_GPIO_PORTI
  - LL_PWR_GPIO_PORTJ (*) (*) value can be used only for STM32U59xxx and STM32U5A9xx devices.
- **GPIOPin:** This parameter can be one of the following values:
  - LL_PWR_GPIO_PIN_0
  - LL_PWR_GPIO_PIN_1
  - LL_PWR_GPIO_PIN_2
  - LL_PWR_GPIO_PIN_3
  - LL_PWR_GPIO_PIN_4
  - LL_PWR_GPIO_PIN_5
  - LL_PWR_GPIO_PIN_6
  - LL_PWR_GPIO_PIN_7
  - LL_PWR_GPIO_PIN_8
  - LL_PWR_GPIO_PIN_9
  - LL_PWR_GPIO_PIN_10
  - LL_PWR_GPIO_PIN_11
  - LL_PWR_GPIO_PIN_12
  - LL_PWR_GPIO_PIN_13
  - LL_PWR_GPIO_PIN_14
  - LL_PWR_GPIO_PIN_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PUCRx PUx LL_PWR_IsEnabledGPIOPullUp

LL_PWR_EnableGPIOPullDown

Function name

__STATIC_INLINE void LL_PWR_EnableGPIOPullDown (uint32_t GPIOPort, uint32_t GPIOPin)

Function description

Enable GPIO pull-down in Standby and Shutdown modes.

Parameters

- **GPIOPort:** This parameter can be one of the following values:
  – LL_PWR_GPIO_PORTA
  – LL_PWR_GPIO_PORTB
  – LL_PWR_GPIO_PORTC
  – LL_PWR_GPIO_PORTD
  – LL_PWR_GPIO_PORTE
  – LL_PWR_GPIO_PORTF
  – LL_PWR_GPIO_PORTG
  – LL_PWR_GPIO_PORTH
  – LL_PWR_GPIO_PORTI
  – LL_PWR_GPIO_PORTJ (*) (*) value can be used only for STM32U59xxx and STM32U5A9xx devices.
- **GPIOPin:** This parameter can be a combination of the following values:
  – LL_PWR_GPIO_PIN_0
  – LL_PWR_GPIO_PIN_1
  – LL_PWR_GPIO_PIN_2
  – LL_PWR_GPIO_PIN_3
  – LL_PWR_GPIO_PIN_4
  – LL_PWR_GPIO_PIN_5
  – LL_PWR_GPIO_PIN_6
  – LL_PWR_GPIO_PIN_7
  – LL_PWR_GPIO_PIN_8
  – LL_PWR_GPIO_PIN_9
  – LL_PWR_GPIO_PIN_10
  – LL_PWR_GPIO_PIN_11
  – LL_PWR_GPIO_PIN_12
  – LL_PWR_GPIO_PIN_13
  – LL_PWR_GPIO_PIN_14
  – LL_PWR_GPIO_PIN_15

Return values

- **None.:**

Reference Manual to LL API cross reference:

- PDCRx PDx LL_PWR_EnableGPIOPullDown

LL_PWR_DisableGPIOPullDown

Function name

__STATIC_INLINE void LL_PWR_DisableGPIOPullDown (uint32_t GPIOPort, uint32_t GPIOPin)

Function description

Disable GPIO pull-down in Standby and Shutdown modes.

Parameters

- **GPIOPort:** This parameter can be one of the following values:
  – LL_PWR_GPIO_PORTA
  – LL_PWR_GPIO_PORTB
  – LL_PWR_GPIO_PORTC
  – LL_PWR_GPIO_PORTD
  – LL_PWR_GPIO_PORTE
  – LL_PWR_GPIO_PORTF
  – LL_PWR_GPIO_PORTG
  – LL_PWR_GPIO_PORTH
  – LL_PWR_GPIO_PORTI
  – LL_PWR_GPIO_PORTJ (*) (*) value can be used only for STM32U59xxx and STM32U5A9xx devices.
- **GPIOPin:** This parameter can be a combination of the following values:
  – LL_PWR_GPIO_PIN_0
  – LL_PWR_GPIO_PIN_1
  – LL_PWR_GPIO_PIN_2
  – LL_PWR_GPIO_PIN_3
  – LL_PWR_GPIO_PIN_4
  – LL_PWR_GPIO_PIN_5
  – LL_PWR_GPIO_PIN_6
  – LL_PWR_GPIO_PIN_7
  – LL_PWR_GPIO_PIN_8
  – LL_PWR_GPIO_PIN_9
  – LL_PWR_GPIO_PIN_10
  – LL_PWR_GPIO_PIN_11
  – LL_PWR_GPIO_PIN_12
  – LL_PWR_GPIO_PIN_13
  – LL_PWR_GPIO_PIN_14
  – LL_PWR_GPIO_PIN_15

Return values

- **None.:**

Reference Manual to LL API cross reference:

- PDCRx PDx LL_PWR_DisableGPIOPullDown

LL_PWR_IsEnabledGPIOPullDown

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullDown (uint32_t GPIOPort, uint32_t GPIOPin)

Function description

Check if GPIO pull-down in Standby and Shutdown modes is enabled.

Parameters

- **GPIOPort:** This parameter can be one of the following values:
  – LL_PWR_GPIO_PORTA
  – LL_PWR_GPIO_PORTB
  – LL_PWR_GPIO_PORTC
  – LL_PWR_GPIO_PORTD
  – LL_PWR_GPIO_PORTE
  – LL_PWR_GPIO_PORTF
  – LL_PWR_GPIO_PORTG
  – LL_PWR_GPIO_PORTH
  – LL_PWR_GPIO_PORTI
  – LL_PWR_GPIO_PORTJ (*) (*) value can be used only for STM32U59xxx and STM32U5A9xx devices.
- **GPIOPin:** This parameter can be one of the following values:
  – LL_PWR_GPIO_PIN_0
  – LL_PWR_GPIO_PIN_1
  – LL_PWR_GPIO_PIN_2
  – LL_PWR_GPIO_PIN_3
  – LL_PWR_GPIO_PIN_4
  – LL_PWR_GPIO_PIN_5
  – LL_PWR_GPIO_PIN_6
  – LL_PWR_GPIO_PIN_7
  – LL_PWR_GPIO_PIN_8
  – LL_PWR_GPIO_PIN_9
  – LL_PWR_GPIO_PIN_10
  – LL_PWR_GPIO_PIN_11
  – LL_PWR_GPIO_PIN_12
  – LL_PWR_GPIO_PIN_13
  – LL_PWR_GPIO_PIN_14
  – LL_PWR_GPIO_PIN_15

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PDCRx PDx LL_PWR_IsEnabledGPIOPullUp

LL_PWR_GetRegulCurrentVOS

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRegulCurrentVOS (void)

Function description

Get currently voltage scaling applied to VCORE.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_REGU_VOLTAGE_SCALE1
  – LL_PWR_REGU_VOLTAGE_SCALE2
  – LL_PWR_REGU_VOLTAGE_SCALE3
  – LL_PWR_REGU_VOLTAGE_SCALE4

Reference Manual to LL API cross reference:

- SVMSR ACTVOS[1:0] LL_PWR_GetRegulCurrentVOS

LL_PWR_IsActiveFlag_BOOST

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_BOOST (void)

Function description

Indicate whether the EPOD (Embedded Power Distribution) booster is ready OR not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR BOOSTRDY LL_PWR_IsActiveFlag_BOOST

LL_PWR_IsActiveFlag_USBBOOST

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_USBBOOST (void)

Function description

Indicate whether the USB EPOD (Embedded Power Distribution) booster is ready OR not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR USBBOOSTRDY LL_PWR_IsActiveFlag_USBBOOST

LL_PWR_IsActiveFlag_VOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VOS (void)

Function description

Indicate whether the regulator voltage output is above voltage scaling range or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VOSR VOSRDY LL_PWR_IsActiveFlag_VOS

LL_PWR_IsActiveFlag_SB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)

Function description

Indicate whether the system was in standby mode or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_STOP

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_STOP (void)

Function description

Indicate whether the system was in stop mode or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR STOPF LL_PWR_IsActiveFlag_STOP

LL_PWR_IsActiveFlag_REGULATOR

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGULATOR (void)

Function description

Indicate whether the regulator supply is LDO or SMPS.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR REGS LL_PWR_IsActiveFlag_REGULATOR

LL_PWR_IsActiveFlag_PVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)

Function description

Indicate whether the VDD voltage is below the threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_ACTVOS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_ACTVOS (void)

Function description

Indicate whether the regulator voltage output is equal to current used voltage scaling range or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR ACTVOSRDY LL_PWR_IsActiveFlag_ACTVOS

LL_PWR_IsActiveFlag_VDDUSB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VDDUSB (void)

Function description

Indicate whether the VDDUSB is below the threshold of monitor or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR VDDUSBRDY LL_PWR_IsActiveFlag_VDDUSB

LL_PWR_IsActiveFlag_VDDIO2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VDDIO2 (void)

Function description

Indicate whether the VDDIO2 is below the threshold of monitor or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR VDDIO2RDY LL_PWR_IsActiveFlag_VDDIO2

LL_PWR_IsActiveFlag_VDDA1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VDDA1 (void)

Function description

Indicate whether the VDDA1 is below the threshold of monitor 1 or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR VDDA1RDY LL_PWR_IsActiveFlag_VDDA1

LL_PWR_IsActiveFlag_VDDA2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VDDA2 (void)

Function description

Indicate whether the VDDA2 is below the threshold of monitor 2 or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SVMSR VDDA2RDY LL_PWR_IsActiveFlag_VDDA2

LL_PWR_IsActiveFlag_VBATH

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_VBATH (void)

Function description

Indicate whether the VBAT level is below high threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR VBATH LL_PWR_IsActiveFlag_VBATH

LL_PWR_IsActiveFlag_TEMPL

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_TEMPL (void)

Function description

Indicate whether the CPU temperature level is above low threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR TEMPL LL_PWR_IsActiveFlag_TEMPL

LL_PWR_IsActiveFlag_TEMPH

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_TEMPH (void)

Function description

Indicate whether the CPU temperature level is below high threshold or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDSR TEMPH LL_PWR_IsActiveFlag_TEMPH

LL_PWR_IsActiveFlag_WU1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU1 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 1.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF1 LL_PWR_IsActiveFlag_WU1

LL_PWR_IsActiveFlag_WU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU2 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF2 LL_PWR_IsActiveFlag_WU2

LL_PWR_IsActiveFlag_WU3

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU3 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 3.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF3 LL_PWR_IsActiveFlag_WU3

LL_PWR_IsActiveFlag_WU4

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU4 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 4.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF4 LL_PWR_IsActiveFlag_WU4

LL_PWR_IsActiveFlag_WU5

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU5 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 5.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF5 LL_PWR_IsActiveFlag_WU5

LL_PWR_IsActiveFlag_WU6

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU6 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 6.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF6 LL_PWR_IsActiveFlag_WU6

LL_PWR_IsActiveFlag_WU7

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU7 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 7.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF7 LL_PWR_IsActiveFlag_WU7

LL_PWR_IsActiveFlag_WU8

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU8 (void)

Function description

Indicate whether a wakeup event is detected on wake up pin 8.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- WUSR WUF8 LL_PWR_IsActiveFlag_WU8

LL_PWR_ClearFlag_STOP

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_STOP (void)

Function description

Clear stop flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CSSF LL_PWR_ClearFlag_STOP

LL_PWR_ClearFlag_SB

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_SB (void)

Function description

Clear standby flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CSSF LL_PWR_ClearFlag_SB

LL_PWR_ClearFlag_WU1

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU1 (void)

Function description

Clear wake up flag 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF1 LL_PWR_ClearFlag_WU1

LL_PWR_ClearFlag_WU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU2 (void)

Function description

Clear wake up flag 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF2 LL_PWR_ClearFlag_WU2

LL_PWR_ClearFlag_WU3

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU3 (void)

Function description

Clear wake up flag 3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF3 LL_PWR_ClearFlag_WU3

LL_PWR_ClearFlag_WU4

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU4 (void)

Function description

Clear wake up flag 4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF4 LL_PWR_ClearFlag_WU4

LL_PWR_ClearFlag_WU5

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU5 (void)

Function description

Clear wake up flag 5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF5 LL_PWR_ClearFlag_WU5

LL_PWR_ClearFlag_WU6

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU6 (void)

Function description

Clear wake up flag 6.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF6 LL_PWR_ClearFlag_WU6

LL_PWR_ClearFlag_WU7

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU7 (void)

Function description

Clear wake up flag 7.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF7 LL_PWR_ClearFlag_WU7

LL_PWR_ClearFlag_WU8

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU8 (void)

Function description

Clear wake up flag 8.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF8 LL_PWR_ClearFlag_WU8

LL_PWR_ClearFlag_WU

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)

Function description

Clear all wake up flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WUSCR CWUF LL_PWR_ClearFlag_WU

LL_PWR_EnableNSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_EnableNSecurePrivilege (void)

Function description

Enable privileged mode for nsecure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_PWR_EnableNSecurePrivilege

LL_PWR_DisableNSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_DisableNSecurePrivilege (void)

Function description

Disable privileged mode for nsecure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_PWR_DisableNSecurePrivilege

LL_PWR_IsEnabledNSecurePrivilege

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledNSecurePrivilege (void)

Function description

Check if privileged mode for nsecure items is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_PWR_IsEnabledNSecurePrivilege

LL_PWR_EnableSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_EnableSecurePrivilege (void)

Function description

Enable privileged mode for secure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_PWR_EnableSecurePrivilege

LL_PWR_DisableSecurePrivilege

Function name

__STATIC_INLINE void LL_PWR_DisableSecurePrivilege (void)

Function description

Disable privileged mode for secure items.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_PWR_DisableSecurePrivilege

LL_PWR_IsEnabledSecurePrivilege

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledSecurePrivilege (void)

Function description

Check if privileged mode for secure items is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_PWR_IsEnabledSecurePrivilege

LL_PWR_ConfigSecure

Function name

__STATIC_INLINE void LL_PWR_ConfigSecure (uint32_t SecureConfig)

Function description

Configure secure attribute mode.

Parameters

- **SecureConfig:** This parameter can be the full combination of the following values:
  - LL_PWR_WAKEUP_PIN1_NSEC or LL_PWR_WAKEUP_PIN1_SEC
  - LL_PWR_WAKEUP_PIN2_NSEC or LL_PWR_WAKEUP_PIN2_SEC
  - LL_PWR_WAKEUP_PIN3_NSEC or LL_PWR_WAKEUP_PIN3_SEC
  - LL_PWR_WAKEUP_PIN4_NSEC or LL_PWR_WAKEUP_PIN4_SEC
  - LL_PWR_WAKEUP_PIN5_NSEC or LL_PWR_WAKEUP_PIN5_SEC
  - LL_PWR_WAKEUP_PIN6_NSEC or LL_PWR_WAKEUP_PIN6_SEC
  - LL_PWR_WAKEUP_PIN7_NSEC or LL_PWR_WAKEUP_PIN7_SEC
  - LL_PWR_WAKEUP_PIN8_NSEC or LL_PWR_WAKEUP_PIN8_SEC
  - LL_PWR_LPM_NSEC or LL_PWR_LPM_SEC
  - LL_PWR_VDM_NSEC or LL_PWR_VDM_SEC
  - LL_PWR_VB_NSEC or LL_PWR_VB_SEC
  - LL_PWR_APC_NSEC or LL_PWR_APC_SEC

Return values

- **None.:**

Notes

- This API can be executed only by CPU in secure mode.

Reference Manual to LL API cross reference:

- SECCFGR WUP1SEC LL_PWR_ConfigSecure
- SECCFGR WUP2SEC LL_PWR_ConfigSecure
- SECCFGR WUP3SEC LL_PWR_ConfigSecure
- SECCFGR WUP4SEC LL_PWR_ConfigSecure
- SECCFGR WUP5SEC LL_PWR_ConfigSecure
- SECCFGR WUP6SEC LL_PWR_ConfigSecure
- SECCFGR WUP7SEC LL_PWR_ConfigSecure
- SECCFGR WUP8SEC LL_PWR_ConfigSecure
- SECCFGR LPMSEC LL_PWR_ConfigSecure
- SECCFGR VDMSEC LL_PWR_ConfigSecure
- SECCFGR VBSEC LL_PWR_ConfigSecure
- SECCFGR APCSEC LL_PWR_ConfigSecure

LL_PWR_GetConfigSecure

Function name

__STATIC_INLINE uint32_t LL_PWR_GetConfigSecure (void)

Function description

Get secure attribute configuration.

Return values

- **Returned:** value is the combination of the following values:
  – LL_PWR_WAKEUP_PIN1_NSEC or LL_PWR_WAKEUP_PIN1_SEC
  – LL_PWR_WAKEUP_PIN2_NSEC or LL_PWR_WAKEUP_PIN2_SEC
  – LL_PWR_WAKEUP_PIN3_NSEC or LL_PWR_WAKEUP_PIN3_SEC
  – LL_PWR_WAKEUP_PIN4_NSEC or LL_PWR_WAKEUP_PIN4_SEC
  – LL_PWR_WAKEUP_PIN5_NSEC or LL_PWR_WAKEUP_PIN5_SEC
  – LL_PWR_WAKEUP_PIN6_NSEC or LL_PWR_WAKEUP_PIN6_SEC
  – LL_PWR_WAKEUP_PIN7_NSEC or LL_PWR_WAKEUP_PIN7_SEC
  – LL_PWR_WAKEUP_PIN8_NSEC or LL_PWR_WAKEUP_PIN8_SEC
  – LL_PWR_LPM_NSEC or LL_PWR_LPM_SEC
  – LL_PWR_VDM_NSEC or LL_PWR_VDM_SEC
  – LL_PWR_VB_NSEC or LL_PWR_VB_SEC
  – LL_PWR_APC_NSEC or LL_PWR_APC_SEC

Notes

- This API can be executed only by CPU in secure mode.

Reference Manual to LL API cross reference:

- SECCFGR WUP1SEC LL_PWR_GetConfigSecure
- SECCFGR WUP2SEC LL_PWR_GetConfigSecure
- SECCFGR WUP3SEC LL_PWR_GetConfigSecure
- SECCFGR WUP4SEC LL_PWR_GetConfigSecure
- SECCFGR WUP5SEC LL_PWR_GetConfigSecure
- SECCFGR WUP6SEC LL_PWR_GetConfigSecure
- SECCFGR WUP7SEC LL_PWR_GetConfigSecure
- SECCFGR WUP8SEC LL_PWR_GetConfigSecure
- SECCFGR LPMSEC LL_PWR_GetConfigSecure
- SECCFGR VDMSEC LL_PWR_GetConfigSecure
- SECCFGR VBSEC LL_PWR_GetConfigSecure
- SECCFGR APCSEC LL_PWR_GetConfigSecure

LL_PWR_DeInit

Function name

ErrorStatus LL_PWR_DeInit (void)

Function description

De-initialize the PWR registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS : PWR registers are de-initialized.
  – ERROR : not applicable.

113.2 PWR Firmware driver defines

The following section lists the various define and macros of the module.

113.2.1 PWR

PWR

PWR VBAT Charging Resistor Selection

LL_PWR_BATT_CHARG_RESISTOR_5K

Charge the battery through a 5 kO resistor

LL_PWR_BATT_CHARG_RESISTOR_1_5K

Charge the battery through a 1.5 kO resistor

Clear Flags Defines

LL_PWR_SR_CSSF

Clear Stop and Standby flags

LL_PWR_WUSCR_CWUF1

Clear Wakeup flag 1

LL_PWR_WUSCR_CWUF2

Clear Wakeup flag 2

LL_PWR_WUSCR_CWUF3

Clear Wakeup flag 3

LL_PWR_WUSCR_CWUF4

Clear Wakeup flag 4

LL_PWR_WUSCR_CWUF5

Clear Wakeup flag 5

LL_PWR_WUSCR_CWUF6

Clear Wakeup flag 6

LL_PWR_WUSCR_CWUF7

Clear Wakeup flag 7

LL_PWR_WUSCR_CWUF8

Clear Wakeup flag 8

LL_PWR_WUSCR_CWUF_ALL

Clear all Wakeup flags

PWR DCACHE1 Content Retention in Stop Mode

LL_PWR_DCACHE1RAM_STOP_NO_RETENTION

DCACHE1 SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_DCACHE1RAM_STOP_FULL_RETENTION

DCACHE1 SRAM retention in Stop mode (Stop 0, 1, 2)

PWR DCACHE2 Content Retention in Stop Mode

LL_PWR_DCACHE2RAM_STOP_NO_RETENTION

DCACHE2 SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_DCACHE2RAM_STOP_FULL_RETENTION

DCACHE2 SRAM retention in Stop mode (Stop 0, 1, 2)

PWR DMA2DRAM Content Retention in Stop Mode

LL_PWR_DMA2DRAM_STOP_NO_RETENTION

DMA2D SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_DMA2DRAM_STOP_FULL_RETENTION

DMA2D SRAM retention in Stop mode (Stop 0, 1, 2)

PWR DSI RAM Content Retention in Stop Mode

LL_PWR_DSIRAM_STOP_NO_RETENTION

DSI SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_DSIRAM_STOP_FULL_RETENTION

DSI SRAM retention in Stop mode (Stop 0, 1, 2)

Get Flags Defines

LL_PWR_FLAG_VOSRDY

Voltage scaling ready flag

LL_PWR_FLAG_BOOSTRDY

VOS EPOD booster ready flag

LL_PWR_FLAG_USBBOOSTRDY

USB EPOD booster ready flag

LL_PWR_FLAG_STOPF

Stop flag

LL_PWR_FLAG_SBF

Standby flag

LL_PWR_FLAG_VDDA2RDY

VDDA ready flag (versus 1.8 V threshold)

LL_PWR_FLAG_VDDA1RDY

VDDA ready flag (versus 1.6 V threshold)

LL_PWR_FLAG_VDDIO2RDY

VDDIO2 ready flag

LL_PWR_FLAG_VDDUSBRDY

VDDUSB ready flag

LL_PWR_FLAG_ACTVOSRDY

Currently applied VOS ready flag

LL_PWR_FLAG_PVDO

VDD voltage detector output flag

LL_PWR_FLAG_REGS

Regulator selection flag

LL_PWR_FLAG_TEMPH

Temperature level flag (versus high threshold)

LL_PWR_FLAG_TEMPL

Temperature level flag (versus low threshold)

LL_PWR_FLAG_VBATH

Backup domain voltage level flag (versus high threshold)

LL_PWR_WAKEUP_FLAG1
Wakeup flag 1

LL_PWR_WAKEUP_FLAG2
Wakeup flag 2

LL_PWR_WAKEUP_FLAG3
Wakeup flag 3

LL_PWR_WAKEUP_FLAG4
Wakeup flag 4

LL_PWR_WAKEUP_FLAG5
Wakeup flag 5

LL_PWR_WAKEUP_FLAG6
Wakeup flag 6

LL_PWR_WAKEUP_FLAG7
Wakeup flag 7

LL_PWR_WAKEUP_FLAG8
Wakeup flag 8

PWR GPIO Pin Mask

LL_PWR_GPIO_PIN_0
GPIO port I/O pin 0

LL_PWR_GPIO_PIN_1
GPIO port I/O pin 1

LL_PWR_GPIO_PIN_2
GPIO port I/O pin 2

LL_PWR_GPIO_PIN_3
GPIO port I/O pin 3

LL_PWR_GPIO_PIN_4
GPIO port I/O pin 4

LL_PWR_GPIO_PIN_5
GPIO port I/O pin 5

LL_PWR_GPIO_PIN_6
GPIO port I/O pin 6

LL_PWR_GPIO_PIN_7
GPIO port I/O pin 7

LL_PWR_GPIO_PIN_8
GPIO port I/O pin 8

LL_PWR_GPIO_PIN_9
GPIO port I/O pin 9

LL_PWR_GPIO_PIN_10
GPIO port I/O pin 10

LL_PWR_GPIO_PIN_11

GPIO port I/O pin 11

LL_PWR_GPIO_PIN_12

GPIO port I/O pin 12

LL_PWR_GPIO_PIN_13

GPIO port I/O pin 13

LL_PWR_GPIO_PIN_14

GPIO port I/O pin 14

LL_PWR_GPIO_PIN_15

GPIO port I/O pin 15

PWR GPIO Port Selection

LL_PWR_GPIO_PORTA

GPIO port A

LL_PWR_GPIO_PORTB

GPIO port B

LL_PWR_GPIO_PORTC

GPIO port C

LL_PWR_GPIO_PORTD

GPIO port D

LL_PWR_GPIO_PORTE

GPIO port E

LL_PWR_GPIO_PORTF

GPIO port F

LL_PWR_GPIO_PORTG

GPIO port G

LL_PWR_GPIO_PORTH

GPIO port H

LL_PWR_GPIO_PORTI

GPIO port I

LL_PWR_GPIO_PORTJ

GPIO port J

PWR GPRAM Content Retention in Stop Mode

LL_PWR_GRAPHICPERIPHRAM_STOP_NO_RETENTION

Graphic peripherals (LTDC, GFXMMU) SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_GRAPHICPERIPHRAM_STOP_FULL_RETENTION

Graphic peripherals (LTDC, GFXMMU) SRAM retention in Stop mode (Stop 0, 1, 2)

PWR ICACHE Content Retention in Stop Mode

LL_PWR_ICACHERAM_STOP_NO_RETENTION

ICACHE SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_ICACHERAM_STOP_FULL_RETENTION

ICACHE SRAM retention in Stop mode (Stop 0, 1, 2)

PWR Items Secure Attribute

LL_PWR_WAKEUP_PIN1_NSEC

Wake up pin 1 nsecure mode

LL_PWR_WAKEUP_PIN1_SEC

Wake up pin 1 secure mode

LL_PWR_WAKEUP_PIN2_NSEC

Wake up pin 2 nsecure mode

LL_PWR_WAKEUP_PIN2_SEC

Wake up pin 2 secure mode

LL_PWR_WAKEUP_PIN3_NSEC

Wake up pin 3 nsecure mode

LL_PWR_WAKEUP_PIN3_SEC

Wake up pin 3 secure mode

LL_PWR_WAKEUP_PIN4_NSEC

Wake up pin 4 nsecure mode

LL_PWR_WAKEUP_PIN4_SEC

Wake up pin 4 secure mode

LL_PWR_WAKEUP_PIN5_NSEC

Wake up pin 5 nsecure mode

LL_PWR_WAKEUP_PIN5_SEC

Wake up pin 5 secure mode

LL_PWR_WAKEUP_PIN6_NSEC

Wake up pin 6 nsecure mode

LL_PWR_WAKEUP_PIN6_SEC

Wake up pin 6 secure mode

LL_PWR_WAKEUP_PIN7_NSEC

Wake up pin 7 nsecure mode

LL_PWR_WAKEUP_PIN7_SEC

Wake up pin 7 secure mode

LL_PWR_WAKEUP_PIN8_NSEC

Wake up pin 8 nsecure mode

LL_PWR_WAKEUP_PIN8_SEC

Wake up pin 8 secure mode

LL_PWR_LPM_NSEC

Low-power modes nsecure mode

LL_PWR_LPM_SEC

Low-power modes secure mode

LL_PWR_VDM_NSEC

Voltage detection and monitoring nsecure mode

LL_PWR_VDM_SEC

Voltage detection and monitoring secure mode

LL_PWR_VB_NSEC

Backup domain nsecure mode

LL_PWR_VB_SEC

Backup domain secure mode

LL_PWR_APC_NSEC

Pull-up/pull-down nsecure mode

LL_PWR_APC_SEC

Pull-up/pull-down secure mode

PWR JPEG RAM Content Retention in Stop Mode

LL_PWR_JPEGRAM_STOP_NO_RETENTION

JPEG SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_JPEGRAM_STOP_FULL_RETENTION

JPEG SRAM retention in Stop mode (Stop 0, 1, 2)

Low Power Mode Selection

LL_PWR_STOP0_MODE

Stop 0 mode

LL_PWR_STOP1_MODE

Stop 1 mode

LL_PWR_STOP2_MODE

Stop 2 mode

LL_PWR_STOP3_MODE

Stop 3 mode

LL_PWR_STANDBY_MODE

Standby mode

LL_PWR_SHUTDOWN_MODE

Shutdown mode

PWR PERIPHRAM Content Retention in Stop Mode

LL_PWR_PERIPHRAM_STOP_NO_RETENTION

FMAC, FDCAN and USB SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_PERIPHRAM_STOP_FULL_RETENTION

FMAC, FDCAN and USB SRAM retention in Stop mode (Stop 0, 1, 2)

PWR PKARAM Content Retention in Stop Mode

LL_PWR_PKARAM_STOP_NO_RETENTION

PKA32 SRAM no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_PKARAM_STOP_FULL_RETENTION

PKA32 SRAM retention in Stop mode (Stop 0, 1, 2)

PWR Power Voltage Detector Level Selection

LL_PWR_PVDLEVEL_0

Voltage threshold detected by PVD 2.0 V

LL_PWR_PVDLEVEL_1

Voltage threshold detected by PVD 2.2 V

LL_PWR_PVDLEVEL_2

Voltage threshold detected by PVD 2.4 V

LL_PWR_PVDLEVEL_3

Voltage threshold detected by PVD 2.5 V

LL_PWR_PVDLEVEL_4

Voltage threshold detected by PVD 2.6 V

LL_PWR_PVDLEVEL_5

Voltage threshold detected by PVD 2.8 V

LL_PWR_PVDLEVEL_6

Voltage threshold detected by PVD 2.9 V

LL_PWR_PVDLEVEL_7

External input analog voltage on PVD_IN pin, compared to internal VREFINT level

PWR Regulator Supply Selection

LL_PWR_LDO_SUPPLY

LDO regulator supply

LL_PWR_SMPS_SUPPLY

SMPS regulator supply

PWR SRAM1 Content Retention in Run Mode

LL_PWR_SRAM1_RUN_NO_RETENTION

SRAM1 no retention in Run mode

LL_PWR_SRAM1_RUN_FULL_RETENTION

SRAM1 retention in Run mode

PWR SRAM1 Content Retention in Stop Mode

LL_PWR_SRAM1_STOP_NO_RETENTION

SRAM1 no retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE1_RETENTION

SRAM1 page 1 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE2_RETENTION

SRAM1 page 2 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE3_RETENTION

SRAM1 page 3 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE4_RETENTION

SRAM1 page 4 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE5_RETENTION

SRAM1 page 5 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE6_RETENTION

SRAM1 page 6 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE7_RETENTION

SRAM1 page 7 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE8_RETENTION

SRAM1 page 8 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE9_RETENTION

SRAM1 page 9 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE10_RETENTION

SRAM1 page 10 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE11_RETENTION

SRAM1 page 11 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_PAGE12_RETENTION

SRAM1 page 12 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_1_3_RETENTION

SRAM1 pages (1 to 3) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM1_STOP_4_12_RETENTION

SRAM1 pages (4 to 12) retention in Stop mode (Stop 0, 1, 2, 3)

PWR SRAM2 Content Retention in Run Mode

LL_PWR_SRAM2_RUN_NO_RETENTION

SRAM2 no retention in Run mode

LL_PWR_SRAM2_RUN_FULL_RETENTION

SRAM2 retention in Run mode

PWR SRAM2 Content Retention in Standby Mode

LL_PWR_SRAM2_SB_NO_RETENTION

SRAM2 no retention in Stop 3 and Standby mode

LL_PWR_SRAM2_SB_PAGE1_RETENTION

SRAM2 page 1 (8 KB) retention in Stop 3 and Standby mode

LL_PWR_SRAM2_SB_PAGE2_RETENTION

SRAM2 page 2 (54 KB) retention in Stop 3 and Standby mode

LL_PWR_SRAM2_SB_FULL_RETENTION

SRAM2 all pages retention in Stop 3 and Standby mode

PWR SRAM2 Content Retention in Stop Mode

LL_PWR_SRAM2_STOP_NO_RETENTION

SRAM2 no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_SRAM2_STOP_PAGE1_RETENTION

SRAM2 page 1 (8 KB) retention in Stop mode (Stop 0, 1, 2)

LL_PWR_SRAM2_STOP_PAGE2_RETENTION

SRAM2 page 2 (54 KB) retention in Stop mode (Stop 0, 1, 2)

LL_PWR_SRAM2_STOP_FULL_RETENTION

SRAM2 all pages retention in Stop mode (Stop 0, 1, 2)

PWR SRAM3 Content Retention in Run Mode

LL_PWR_SRAM3_RUN_NO_RETENTION

SRAM3 no retention in Run mode

LL_PWR_SRAM3_RUN_FULL_RETENTION

SRAM3 retention in Run mode

PWR SRAM3 Content Retention in Stop Mode

LL_PWR_SRAM3_STOP_NO_RETENTION

SRAM3 no retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE1_RETENTION

SRAM3 page 1 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE2_RETENTION

SRAM3 page 2 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE3_RETENTION

SRAM3 page 3 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE4_RETENTION

SRAM3 page 4 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE5_RETENTION

SRAM3 page 5 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE6_RETENTION

SRAM3 page 6 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE7_RETENTION

SRAM3 page 7 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE8_RETENTION

SRAM3 page 8 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE9_RETENTION

SRAM3 page 9 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE10_RETENTION

SRAM3 page 10 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE11_RETENTION
SRAM3 page 11 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE12_RETENTION
SRAM3 page 12 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_PAGE13_RETENTION
SRAM3 page 13 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_1_8_RETENTION
SRAM3 pages (1 to 8) retention in Stop modes (Stop 0, 1, 2, 3)

LL_PWR_SRAM3_STOP_9_13_RETENTION
SRAM3 pages (9 to 13) retention in Stop modes (Stop 0, 1, 2, 3)

PWR SRAM4 Content Retention in Run Mode

LL_PWR_SRAM4_RUN_NO_RETENTION
SRAM4 no retention in Run mode

LL_PWR_SRAM4_RUN_FULL_RETENTION
SRAM4 retention in Run mode

PWR SRAM4 Content Retention in Stop Mode

LL_PWR_SRAM4_STOP_NO_RETENTION
SRAM4 no retention in Stop mode (Stop 0, 1, 2)

LL_PWR_SRAM4_STOP_FULL_RETENTION
SRAM4 retention in Stop mode (Stop 0, 1, 2)

PWR SRAM5 Content Retention in Run Mode

LL_PWR_SRAM5_RUN_NO_RETENTION
SRAM5 no retention in Run mode

LL_PWR_SRAM5_RUN_FULL_RETENTION
SRAM5 retention in Run mode

PWR SRAM5 Content Retention in Stop Mode

LL_PWR_SRAM5_STOP_NO_RETENTION
SRAM5 no retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE1_RETENTION
SRAM5 page 1 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE2_RETENTION
SRAM5 page 2 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE3_RETENTION
SRAM5 page 3 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE4_RETENTION
SRAM5 page 4 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE5_RETENTION
SRAM5 page 5 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE6_RETENTION
SRAM5 page 6 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE7_RETENTION
SRAM5 page 7 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE8_RETENTION
SRAM5 page 8 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE9_RETENTION
SRAM5 page 4 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE10_RETENTION
SRAM5 page 5 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE11_RETENTION
SRAM5 page 6 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE12_RETENTION
SRAM5 page 7 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_PAGE13_RETENTION
SRAM5 page 8 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM5_STOP_FULL_RETENTION
SRAM5 pages (1 to 13) retention in Stop modes (Stop 0, 1, 2, 3)

PWR SRAM6 Content Retention in Run Mode

LL_PWR_SRAM6_RUN_NO_RETENTION
SRAM6 no retention in Run mode

LL_PWR_SRAM6_RUN_FULL_RETENTION
SRAM6 retention in Run mode

PWR SRAM6 Content Retention in Stop Mode

LL_PWR_SRAM6_STOP_NO_RETENTION
SRAM6 no retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE1_RETENTION
SRAM6 page 1 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE2_RETENTION
SRAM6 page 2 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE3_RETENTION
SRAM6 page 3 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE4_RETENTION
SRAM6 page 4 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE5_RETENTION
SRAM6 page 5 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE6_RETENTION
SRAM6 page 6 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE7_RETENTION

SRAM6 page 7 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_PAGE8_RETENTION

SRAM6 page 8 (64 KB) retention in Stop mode (Stop 0, 1, 2, 3)

LL_PWR_SRAM6_STOP_FULL_RETENTION

SRAM6 pages (1 to 8) retention in Stop modes (Stop 0, 1, 2, 3)

PWR Smart Run Domain Mode

LL_PWR_SRD_STOP_MODE

SmartRun domain AHB3 and APB3 clocks disabled by default in Stop mode (Stop 0, 1, 2)

LL_PWR_SRD_RUN_MODE

SmartRun domain AHB3 and APB3 clocks kept enabled in Stop mode (Stop 0, 1, 2)

PWR Voltage scaling range selection

LL_PWR_REGU_VOLTAGE_SCALE1

Voltage scaling range 1

LL_PWR_REGU_VOLTAGE_SCALE2

Voltage scaling range 2

LL_PWR_REGU_VOLTAGE_SCALE3

Voltage scaling range 3

LL_PWR_REGU_VOLTAGE_SCALE4

Voltage scaling range 4

PWR Wake Up Pin

LL_PWR_WAKEUP_PIN1

Wakeup pin 1 enable

LL_PWR_WAKEUP_PIN2

Wakeup pin 2 enable

LL_PWR_WAKEUP_PIN3

Wakeup pin 3 enable

LL_PWR_WAKEUP_PIN4

Wakeup pin 4 enable

LL_PWR_WAKEUP_PIN5

Wakeup pin 5 enable

LL_PWR_WAKEUP_PIN6

Wakeup pin 6 enable

LL_PWR_WAKEUP_PIN7

Wakeup pin 7 enable

LL_PWR_WAKEUP_PIN8

Wakeup pin 8 enable

PWR Wakeup Pin Selection

LL_PWR_WAKEUP_PIN_SELECTION_0

Wakeup pin selection 0

LL_PWR_WAKEUP_PIN_SELECTION_1

Wakeup pin selection 1

LL_PWR_WAKEUP_PIN_SELECTION_2

Wakeup pin selection 2

LL_PWR_WAKEUP_PIN_SELECTION_3

Wakeup pin selection 3

PWR Configuration

LL_PWR_EnableVDDUSB

LL_PWR_DisableVDDUSB

LL_PWR_IsEnabledVDDUSB

LL_PWR_EnableVDDIO2

LL_PWR_DisableVDDIO2

LL_PWR_IsEnabledVDDIO2

LL_PWR_EnableVDDA

LL_PWR_DisableVDDA

LL_PWR_IsEnabledVDDA

LL_PWR_EnableVDDUSBMonitor

LL_PWR_DisableVDDUSBMonitor

LL_PWR_IsEnabledVDDUSBMonitor

LL_PWR_EnableVDDIO2Monitor

LL_PWR_DisableVDDIO2Monitor

LL_PWR_IsEnabledVDDIO2Monitor

LL_PWR_EnableVDDAMonitor1

LL_PWR_DisableVDDAMonitor1

LL_PWR_IsEnabledVDDAMonitor1

LL_PWR_EnableVDDAMonitor2

LL_PWR_DisableVDDAMonitor2

LL_PWR_IsEnabledVDDAMonitor2

Common Write and Read Registers Macros

LL_PWR_WriteReg

Description:

- Write a value in PWR register.

Parameters:

- __REG__: Register to be written.
- __VALUE__: Value to be written in the register.

Return value:

- None.

LL_PWR_ReadReg

Description:

- Read a value in PWR register.

Parameters:

- __REG__: Register to be read.

Return value:

- Register: value.

114 LL RCC Generic Driver

114.1 RCC Firmware driver registers structures

114.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32u5xx_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_Frequency***
- ***uint32_t PCLK2_Frequency***
- ***uint32_t PCLK3_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::HCLK_Frequency***
  HCLK clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK1_Frequency***
  PCLK1 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK2_Frequency***
  PCLK2 clock frequency
- ***uint32_t LL_RCC_ClocksTypeDef::PCLK3_Frequency***
  PCLK3 clock frequency

114.2 RCC Firmware driver API description

The following section lists the various functions of the RCC library.

114.2.1 Detailed description of functions

LL_RCC_HSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableCSS (void)

Function description

Enable the Clock Security System.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CSSON LL_RCC_HSE_EnableCSS

LL_RCC_HSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_EnableBypass (void)

Function description

Enable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_EnableBypass

LL_RCC_HSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_HSE_DisableBypass (void)

Function description

Disable HSE external oscillator (HSE Bypass)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEBYP LL_RCC_HSE_DisableBypass

LL_RCC_HSE_Enable

Function name

__STATIC_INLINE void LL_RCC_HSE_Enable (void)

Function description

Enable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Enable

LL_RCC_HSE_Disable

Function name

__STATIC_INLINE void LL_RCC_HSE_Disable (void)

Function description

Disable HSE crystal oscillator (HSE ON)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_Disable

LL_RCC_HSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)

Function description

Check if HSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSE_SetClockMode

Function name

__STATIC_INLINE void LL_RCC_HSE_SetClockMode (uint32_t HSEMode)

Function description

Set external HSE clock mode.

Parameters

- **HSEMode:** This parameter can be one of the following values:
  - LL_RCC_HSE_ANALOG_MODE
  - LL_RCC_HSE_DIGITAL_MODE

Return values

- **None:**

Notes

- This bit can be written only if the HSE oscillator is disabled

Reference Manual to LL API cross reference:

- CR HSEEXT LL_RCC_HSE_SetClockMode

LL_RCC_HSE_GetClockMode

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_GetClockMode (void)

Function description

Get External HSE clock mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_HSE_ANALOG_MODE
  - LL_RCC_HSE_DIGITAL_MODE

Reference Manual to LL API cross reference:

- CR HSEEXT LL_RCC_HSE_GetClockMode

LL_RCC_HSI_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_EnableInStopMode (void)

Function description

Enable HSI even in stop mode.

Return values

- **None:**

Notes

- HSI oscillator is forced ON even in Stop mode

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_EnableInStopMode

LL_RCC_HSI_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_HSI_DisableInStopMode (void)

Function description

Disable HSI in stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_DisableInStopMode

LL_RCC_HSI_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsEnabledInStopMode (void)

Function description

Check if HSI is enabled in stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_IsEnabledInStopMode

LL_RCC_HSI_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI_Enable (void)

Function description

Enable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI_Disable (void)

Function description

Disable HSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSION LL_RCC_HSI_Disable

LL_RCC_HSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_IsReady (void)

Function description

Check if HSI clock is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIRDY LL_RCC_HSI_IsReady

LL_RCC_HSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibration (void)

Function description

Get HSI Calibration value.

Return values

- **Between:** Min_Data = 0 and Max_Data = 127

Notes

- When HSITRIM is written, HSICAL is updated with the sum of HSITRIM and the factory trim value

Reference Manual to LL API cross reference:

- ICSCR3 HSICAL LL_RCC_HSI_GetCalibration

LL_RCC_HSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_HSI_SetCalibTrimming (uint32_t Value)

Function description

Set HSI Calibration trimming.

Parameters

- **Value:** Between Min_Data = 0 and Max_Data = 127

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the HSICAL
- Default value is 64, which, when added to the HSICAL value, should trim the HSI to 16 MHz +/- 1 %

Reference Manual to LL API cross reference:

- ICSCR3 HSITRIM LL_RCC_HSI_SetCalibTrimming

LL_RCC_HSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetCalibTrimming (void)

Function description

Get HSI Calibration trimming.

Return values

- **Between:** Min_Data = 0 and Max_Data = 127

Reference Manual to LL API cross reference:

- ICSC3R HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_HSI48_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Enable (void)

Function description

Enable HSI48.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSI48ON LL_RCC_HSI48_Enable

LL_RCC_HSI48_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI48_Disable (void)

Function description

Disable HSI48.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSI48ON LL_RCC_HSI48_Disable

LL_RCC_HSI48_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_IsReady (void)

Function description

Check if HSI48 oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSI48RDY LL_RCC_HSI48_IsReady

LL_RCC_HSI48_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI48_GetCalibration (void)

Function description

Get HSI48 Calibration value.

Return values

- **Between:** Min_Data = 0x00 and Max_Data = 0x1FF

Reference Manual to LL API cross reference:

- CRRCR HSI48CAL LL_RCC_HSI48_GetCalibration

LL_RCC_LSE_Enable

Function name

__STATIC_INLINE void LL_RCC_LSE_Enable (void)

Function description

Enable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Enable

LL_RCC_LSE_Disable

Function name

__STATIC_INLINE void LL_RCC_LSE_Disable (void)

Function description

Disable Low Speed External (LSE) crystal.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_EnableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableBypass (void)

Function description

Enable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_EnableBypass

LL_RCC_LSE_DisableBypass

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableBypass (void)

Function description

Disable external clock source (LSE bypass).

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_SetDriveCapability

Function name

__STATIC_INLINE void LL_RCC_LSE_SetDriveCapability (uint32_t LSEDrive)

Function description

Set LSE oscillator drive capability.

Parameters

- **LSEDrive:** This parameter can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Return values

- **None:**

Notes

- The oscillator is in Xtal mode when it is not in bypass mode.

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_SetDriveCapability

LL_RCC_LSE_GetDriveCapability

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_GetDriveCapability (void)

Function description

Get LSE oscillator drive capability.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LSEDRIVE_LOW
  - LL_RCC_LSEDRIVE_MEDIUMLOW
  - LL_RCC_LSEDRIVE_MEDIUMHIGH
  - LL_RCC_LSEDRIVE_HIGH

Reference Manual to LL API cross reference:

- BDCR LSEDRV LL_RCC_LSE_GetDriveCapability

LL_RCC_LSE_EnableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableCSS (void)

Function description

Enable Clock security system on LSE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_EnableCSS

LL_RCC_LSE_DisableCSS

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableCSS (void)

Function description

Disable Clock security system on LSE.

Return values

- **None:**

Notes

- Clock security system can be disabled only after a LSE failure detection. In that case it MUST be disabled by software.

Reference Manual to LL API cross reference:

- BDCR LSECSSON LL_RCC_LSE_DisableCSS

LL_RCC_LSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsReady (void)

Function description

Check if LSE oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSERDY LL_RCC_LSE_IsReady

LL_RCC_LSE_EnablePropagation

Function name

__STATIC_INLINE void LL_RCC_LSE_EnablePropagation (void)

Function description

Enable LSE oscillator propagation for system clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSESYSEN LL_RCC_LSE_EnablePropagation

LL_RCC_LSESYS_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSESYS_IsReady (void)

Function description

Check if LSESYS oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSESYSRDY LL_RCC_LSESYS_IsReady

LL_RCC_LSE_DisablePropagation

Function name

__STATIC_INLINE void LL_RCC_LSE_DisablePropagation (void)

Function description

Disable LSE oscillator propagation for system clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSESYSEN LL_RCC_LSE_DisablePropagation

LL_RCC_LSE_IsPropagationReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsPropagationReady (void)

Function description

Check if LSE oscillator propagation for system clock Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSESYSRDY LL_RCC_LSE_IsPropagationReady

LL_RCC_LSE_IsCSSDetected

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsCSSDetected (void)

Function description

Check if CSS on LSE failure Detection.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSECSSD LL_RCC_LSE_IsCSSDetected

LL_RCC_LSE_EnableGlitchFilter

Function name

__STATIC_INLINE void LL_RCC_LSE_EnableGlitchFilter (void)

Function description

Enable LSE clock glitch filter.

Return values

- **None:**

Notes

- The glitches on LSE can be filtred by setting the LSEGFON.
- LSEGFON must be written when the LSE is disabled (LSEON = 0 and LSERDY = 0).

Reference Manual to LL API cross reference:

- BDCR LSEGFON LL_RCC_LSE_EnableGlitchFilter

LL_RCC_LSE_DisableGlitchFilter

Function name

__STATIC_INLINE void LL_RCC_LSE_DisableGlitchFilter (void)

Function description

Disable LSE clock glitch filter.

Return values

- **None:**

Notes

- LSEGFON must be written when the LSE is disabled (LSEON = 0 and LSERDY = 0).

Reference Manual to LL API cross reference:

- BDCR LSEGFON LL_RCC_LSE_DisableGlitchFilter

LL_RCC_LSE_IsGlitchFilterEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsGlitchFilterEnabled (void)

Function description

Check if LSE clock glitch filter is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSEGFON LL_RCC_LSE_IsGlitchFilterEnabled

LL_RCC_LSI_Enable

Function name

__STATIC_INLINE void LL_RCC_LSI_Enable (void)

Function description

Enable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSION LL_RCC_LSI_Enable

LL_RCC_LSI_Disable

Function name

__STATIC_INLINE void LL_RCC_LSI_Disable (void)

Function description

Disable LSI Oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsReady (void)

Function description

Check if LSI is Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_LSI_SetPrescaler

Function name

__STATIC_INLINE void LL_RCC_LSI_SetPrescaler (uint32_t LSIPrescaler)

Function description

Set LSI prescaler.

Parameters

- **LSIPrescaler:** This parameter can be one of the following values:
  – LL_RCC_LSI_DIV_1
  – LL_RCC_LSI_DIV_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSIPREDIV LL_RCC_LSI_SetPrescaler

LL_RCC_LSI_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_GetPrescaler (void)

Function description

Get LSI prescaler.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_LSI_DIV_1
  – LL_RCC_LSI_DIV_128

Reference Manual to LL API cross reference:

- BDCR LSIPREDIV LL_RCC_LSI_GetPrescaler

LL_RCC_MSIK_Enable

Function name

__STATIC_INLINE void LL_RCC_MSIK_Enable (void)

Function description

Enable MSIK oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSIKON LL_RCC_MSIK_Enable

LL_RCC_MSIK_Disable

Function name

__STATIC_INLINE void LL_RCC_MSIK_Disable (void)

Function description

Disable MSIK oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSIKON LL_RCC_MSIK_Disable

LL_RCC_MSIK_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIK_IsReady (void)

Function description

Check if MSIK oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR MSIKRDY LL_RCC_MSIK_IsReady

LL_RCC_SHSI_Enable

Function name

__STATIC_INLINE void LL_RCC_SHSI_Enable (void)

Function description

Enable SHSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SHSION LL_RCC_SHSI_Enable

LL_RCC_SHSI_Disable

Function name

__STATIC_INLINE void LL_RCC_SHSI_Disable (void)

Function description

Disable SHSI oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SHSION LL_RCC_SHSI_Disable

LL_RCC_SHSI_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_SHSI_IsReady (void)

Function description

Check if SHSI oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR SHSIRDY LL_RCC_SHSI_IsReady

LL_RCC_MSIS_Enable

Function name

__STATIC_INLINE void LL_RCC_MSIS_Enable (void)

Function description

Enable MSIS oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSISON LL_RCC_MSIS_Enable

LL_RCC_MSIS_Disable

Function name

__STATIC_INLINE void LL_RCC_MSIS_Disable (void)

Function description

Disable MSIS oscillator.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSISON LL_RCC_MSIS_Disable

LL_RCC_MSIS_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIS_IsReady (void)

Function description

Check if MSIS oscillator Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR MSISRDY LL_RCC_MSIS_IsReady

LL_RCC_MSI_EnablePLLMode

Function name

__STATIC_INLINE void LL_RCC_MSI_EnablePLLMode (void)

Function description

Enable MSI PLL-mode (Hardware auto calibration with LSE)

Return values

- **None:**

Notes

- MSIPLLEN must be enabled after LSE is enabled (LSEON enabled) and ready (LSERDY set by hardware)
- hardware protection to avoid enabling MSIPLLEN if LSE is not ready

Reference Manual to LL API cross reference:

- CR MSIPLLEN LL_RCC_MSI_EnablePLLMode

LL_RCC_MSI_DisablePLLMode

Function name

__STATIC_INLINE void LL_RCC_MSI_DisablePLLMode (void)

Function description

Disable MSI-PLL mode.

Return values

- **None:**

Notes

- cleared by hardware when LSE is disabled (LSEON = 0) or when the Clock Security System on LSE detects a LSE failure

Reference Manual to LL API cross reference:

- CR MSIPLLEN LL_RCC_MSI_DisablePLLMode

LL_RCC_IsEnabledPLLMode

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledPLLMode (void)

Function description

Check if MSI-PLL mode has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR MSIPLLEN LL_RCC_IsEnabledPLLMode

LL_RCC_SetMSIPLLMode

Function name

__STATIC_INLINE void LL_RCC_SetMSIPLLMode (uint32_t Source)

Function description

Set clock source in PLL mode.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_PLLMODE_MSIS
  – LL_RCC_PLLMODE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSIPLLSEL LL_RCC_SetMSIPLLMode

LL_RCC_GetMSIPLLMode

Function name

__STATIC_INLINE uint32_t LL_RCC_GetMSIPLLMode (void)

Function description

Get Clock source in PLL Mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_PLLMODE_MSIS
  – LL_RCC_PLLMODE_MSIK

Reference Manual to LL API cross reference:

- CR MSIPLLSEL LL_RCC_GetMSIPLLMode

LL_RCC_Enable_MSIPLLFAST

Function name

__STATIC_INLINE void LL_RCC_Enable_MSIPLLFAST (void)

Function description

Enable MSI fast mode.

Notes

- This bit is used only if PLL mode is selected.

Reference Manual to LL API cross reference:

- CR MSIPLLFAST LL_RCC_Enable_MSIPLLFAST

LL_RCC_Disable_MSIPLLFAST

Function name

__STATIC_INLINE void LL_RCC_Disable_MSIPLLFAST (void)

Function description

Disable MSI fast mode.

Notes

- This bit is used only if PLL mode is selected.

Reference Manual to LL API cross reference:

- CR MSIPLLFAST LL_RCC_Disable_MSIPLLFAST

LL_RCC_MSI_IsEnabledMSIPLLFAST

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_IsEnabledMSIPLLFAST (void)

Function description

Check if MSI PLL Fast Mode is enable.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR MSIPLLFAST LL_RCC_MSI_IsEnabledMSIPLLFAST

LL_RCC_MSI_SetMSIBiasMode

Function name

__STATIC_INLINE void LL_RCC_MSI_SetMSIBiasMode (uint32_t BiasMode)

Function description

Set MSI Bias mode.

Parameters

- **BiasMode:** This parameter can be one of the following values:
  - LL_RCC_MSIBIASMODE_CONTINUOUS
  - LL_RCC_MSIBIASMODE_SAMPLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICSCR1 MSIBIAS LL_RCC_MSI_SetMSIBiasMode

LL_RCC_MSI_GetMSIBiasMode

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_GetMSIBiasMode (void)

Function description

Get MSI Bias mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_MSIBIASMODE_CONTINUOUS
  - LL_RCC_MSIBIASMODE_SAMPLING

Reference Manual to LL API cross reference:

- ICSCR1 MSIBIAS LL_RCC_MSI_GetMSIBiasMode

LL_RCC_MSIK_EnableInStopMode

Function name

__STATIC_INLINE void LL_RCC_MSIK_EnableInStopMode (void)

Function description

Enable MSIK even in stop mode.

Return values

- **None:**

Notes

- MSIK oscillator is forced ON even in Stop mode

Reference Manual to LL API cross reference:

- CR MSIKERON LL_RCC_MSIK_EnableInStopMode

LL_RCC_MSIK_DisableInStopMode

Function name

__STATIC_INLINE void LL_RCC_MSIK_DisableInStopMode (void)

Function description

Disable MSIK in stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR MSIKERON LL_RCC_MSIK_DisableInStopMode

LL_RCC_MSIK_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIK_IsEnabledInStopMode (void)

Function description

Check if MSIK is enabled in stop mode.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR MSIKERON LL_RCC_MSIK_IsEnabledInStopMode

LL_RCC_MSI_EnableRangeSelection

Function name

__STATIC_INLINE void LL_RCC_MSI_EnableRangeSelection (void)

Function description

Enable MSI clock range selection with MSIRANGE register.

Return values

- **None:**

Notes

- Write 0 has no effect. After a standby or a reset MSIRGSEL is at 0 and the MSI range value is provided by MSISRANGE

Reference Manual to LL API cross reference:

- ICSCR1 MSIRGSEL LL_RCC_MSI_EnableRangeSelection

LL_RCC_MSI_IsEnabledRangeSelect

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_IsEnabledRangeSelect (void)

Function description

Check if MSI clock range is selected with MSIRANGE register.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ICSCR1 MSIRGSEL LL_RCC_MSI_IsEnabledRangeSelect

LL_RCC_MSIS_SetRange

Function name

__STATIC_INLINE void LL_RCC_MSIS_SetRange (uint32_t Range)

Function description

Configure the Internal Multi Speed oscillator (MSI) clock range in run mode.

Parameters

- **Range:** This parameter can be one of the following values:
  - LL_RCC_MSISRANGE_0
  - LL_RCC_MSISRANGE_1
  - LL_RCC_MSISRANGE_2
  - LL_RCC_MSISRANGE_3
  - LL_RCC_MSISRANGE_4
  - LL_RCC_MSISRANGE_5
  - LL_RCC_MSISRANGE_6
  - LL_RCC_MSISRANGE_7
  - LL_RCC_MSISRANGE_8
  - LL_RCC_MSISRANGE_9
  - LL_RCC_MSISRANGE_10
  - LL_RCC_MSISRANGE_11
  - LL_RCC_MSISRANGE_12
  - LL_RCC_MSISRANGE_13
  - LL_RCC_MSISRANGE_14
  - LL_RCC_MSISRANGE_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICSCR1 MSISRANGE LL_RCC_MSIS_SetRange

LL_RCC_MSIS_GetRange

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIS_GetRange (void)

Function description

Get the Internal Multi Speed oscillator (MSI) clock range in run mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_MSISRANGE_0
  – LL_RCC_MSISRANGE_1
  – LL_RCC_MSISRANGE_2
  – LL_RCC_MSISRANGE_3
  – LL_RCC_MSISRANGE_4
  – LL_RCC_MSISRANGE_5
  – LL_RCC_MSISRANGE_6
  – LL_RCC_MSISRANGE_7
  – LL_RCC_MSISRANGE_8
  – LL_RCC_MSISRANGE_9
  – LL_RCC_MSISRANGE_10
  – LL_RCC_MSISRANGE_11
  – LL_RCC_MSISRANGE_12
  – LL_RCC_MSISRANGE_13
  – LL_RCC_MSISRANGE_14
  – LL_RCC_MSISRANGE_15

Reference Manual to LL API cross reference:

- ICSCR1 MSISRANGE LL_RCC_MSIS_GetRange

LL_RCC_MSIS_SetRangeAfterStandby

Function name

__STATIC_INLINE void LL_RCC_MSIS_SetRangeAfterStandby (uint32_t Range)

Function description

Configure MSIS range used after standby.

Parameters

- **Range:** This parameter can be one of the following values:
  – LL_RCC_MSISSRANGE_4
  – LL_RCC_MSISSRANGE_5
  – LL_RCC_MSISSRANGE_6
  – LL_RCC_MSISSRANGE_7
  – LL_RCC_MSISSRANGE_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR MSISSRANGE LL_RCC_MSIS_SetRangeAfterStandby

LL_RCC_MSIS_GetRangeAfterStandby

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIS_GetRangeAfterStandby (void)

Function description

Get MSIS range used after standby.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_MSISSRANGE_4
  – LL_RCC_MSISSRANGE_5
  – LL_RCC_MSISSRANGE_6
  – LL_RCC_MSISSRANGE_7
  – LL_RCC_MSISSRANGE_8

Reference Manual to LL API cross reference:

- CSR MSISSRANGE LL_RCC_MSIS_GetRangeAfterStandby

LL_RCC_MSI_SetCalibTrimming

Function name

__STATIC_INLINE void LL_RCC_MSI_SetCalibTrimming (uint32_t Value, uint32_t Oscillator)

Function description

Set MSI OSCILLATORx Calibration trimming.

Parameters

- **Value:** Between Min_Data = 0 and Max_Data = 31
- **Oscillator:** This parameter can be one of the following values:
  – LL_RCC_MSI_OSCILLATOR_0
  – LL_RCC_MSI_OSCILLATOR_1
  – LL_RCC_MSI_OSCILLATOR_2
  – LL_RCC_MSI_OSCILLATOR_3

Return values

- **None:**

Notes

- user-programmable trimming value that is added to the MSICALx

Reference Manual to LL API cross reference:

- ICSCR2 MSITRIMx LL_RCC_MSI_SetCalibTrimming

LL_RCC_MSI_GetCalibTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibTrimming (uint32_t Oscillator)

Function description

Get MSI OSCILLATORx Calibration trimming.

Parameters

- **Oscillator:** This parameter can be one of the following values:
  - LL_RCC_MSI_OSCILLATOR_0
  - LL_RCC_MSI_OSCILLATOR_1
  - LL_RCC_MSI_OSCILLATOR_2
  - LL_RCC_MSI_OSCILLATOR_3

Return values

- **Between:** 0 and 31

Reference Manual to LL API cross reference:

- ICSCR2 MSITRIMx LL_RCC_MSI_GetCalibTrimming

LL_RCC_MSI_GetCalibration

Function name

__STATIC_INLINE uint32_t LL_RCC_MSI_GetCalibration (uint32_t Oscillator)

Function description

Get MSI OSCILLATORx Calibration value.

Parameters

- **Oscillator:** This parameter can be one of the following values:
  - LL_RCC_MSI_OSCILLATOR_0
  - LL_RCC_MSI_OSCILLATOR_1
  - LL_RCC_MSI_OSCILLATOR_2
  - LL_RCC_MSI_OSCILLATOR_3

Return values

- **Between:** Min_Data = 0 and Max_Data = 31

Notes

- When MSITRIMx is written, MSICALx is updated with the sum of MSITRIMx and the factory trim value

Reference Manual to LL API cross reference:

- ICSCR1 MSICALx LL_RCC_MSI_GetCalibration

LL_RCC_MSIK_SetRange

Function name

__STATIC_INLINE void LL_RCC_MSIK_SetRange (uint32_t Range)

Function description

Configure the Internal Multi Speed oscillator (MSIK) clock range in run mode.

Parameters

- **Range:** This parameter can be one of the following values:
  - LL_RCC_MSIKRANGE_0
  - LL_RCC_MSIKRANGE_1
  - LL_RCC_MSIKRANGE_2
  - LL_RCC_MSIKRANGE_3
  - LL_RCC_MSIKRANGE_4
  - LL_RCC_MSIKRANGE_5
  - LL_RCC_MSIKRANGE_6
  - LL_RCC_MSIKRANGE_7
  - LL_RCC_MSIKRANGE_8
  - LL_RCC_MSIKRANGE_9
  - LL_RCC_MSIKRANGE_10
  - LL_RCC_MSIKRANGE_11
  - LL_RCC_MSIKRANGE_12
  - LL_RCC_MSIKRANGE_13
  - LL_RCC_MSIKRANGE_14
  - LL_RCC_MSIKRANGE_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICSCR1 MSIKRANGE LL_RCC_MSIK_SetRange

LL_RCC_MSIK_GetRange

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIK_GetRange (void)

Function description

Get the Internal Multi Speed oscillator (MSIK) clock range in run mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_MSIKRANGE_0
  - LL_RCC_MSIKRANGE_1
  - LL_RCC_MSIKRANGE_2
  - LL_RCC_MSIKRANGE_3
  - LL_RCC_MSIKRANGE_4
  - LL_RCC_MSIKRANGE_5
  - LL_RCC_MSIKRANGE_6
  - LL_RCC_MSIKRANGE_7
  - LL_RCC_MSIKRANGE_8
  - LL_RCC_MSIKRANGE_9
  - LL_RCC_MSIKRANGE_10
  - LL_RCC_MSIKRANGE_11
  - LL_RCC_MSIKRANGE_12
  - LL_RCC_MSIKRANGE_13
  - LL_RCC_MSIKRANGE_14
  - LL_RCC_MSIKRANGE_15

Reference Manual to LL API cross reference:

- ICSCR1 MSIKRANGE LL_RCC_MSIK_GetRange

LL_RCC_MSIK_SetRangeAfterStandby

Function name

__STATIC_INLINE void LL_RCC_MSIK_SetRangeAfterStandby (uint32_t Range)

Function description

Configure MSIK range used after standby.

Parameters

- **Range:** This parameter can be one of the following values:
  - LL_RCC_MSIKSRANGE_4
  - LL_RCC_MSIKSRANGE_5
  - LL_RCC_MSIKSRANGE_6
  - LL_RCC_MSIKSRANGE_7
  - LL_RCC_MSIKSRANGE_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR MSIKSRANGE LL_RCC_MSIK_SetRangeAfterStandby

LL_RCC_MSIK_GetRangeAfterStandby

Function name

__STATIC_INLINE uint32_t LL_RCC_MSIK_GetRangeAfterStandby (void)

Function description

Get MSIK range used after standby.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_MSIKSRANGE_4
  - LL_RCC_MSIKSRANGE_5
  - LL_RCC_MSIKSRANGE_6
  - LL_RCC_MSIKSRANGE_7
  - LL_RCC_MSIKSRANGE_8

Reference Manual to LL API cross reference:

- CSR MSIKSRANGE LL_RCC_MSIK_GetRangeAfterStandby

LL_RCC_LSCO_Enable

Function name

__STATIC_INLINE void LL_RCC_LSCO_Enable (void)

Function description

Enable Low speed clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOEN LL_RCC_LSCO_Enable

LL_RCC_LSCO_Disable

Function name

__STATIC_INLINE void LL_RCC_LSCO_Disable (void)

Function description

Disable Low speed clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOEN LL_RCC_LSCO_Disable

LL_RCC_LSCO_SetSource

Function name

__STATIC_INLINE void LL_RCC_LSCO_SetSource (uint32_t Source)

Function description

Configure Low speed clock selection.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_LSCO_CLKSOURCE_LSI
  – LL_RCC_LSCO_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR LSCOSEL LL_RCC_LSCO_SetSource

LL_RCC_LSCO_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_LSCO_GetSource (void)

Function description

Get Low speed clock selection.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_LSCO_CLKSOURCE_LSI
  – LL_RCC_LSCO_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- BDCR LSCOSEL LL_RCC_LSCO_GetSource

LL_RCC_SetSysClkSource

Function name

__STATIC_INLINE void LL_RCC_SetSysClkSource (uint32_t Source)

Function description

Configure the system clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_MSIS
  - LL_RCC_SYS_CLKSOURCE_HSI
  - LL_RCC_SYS_CLKSOURCE_HSE
  - LL_RCC_SYS_CLKSOURCE_PLL1

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 SW LL_RCC_SetSysClkSource

LL_RCC_GetSysClkSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSysClkSource (void)

Function description

Get the system clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYS_CLKSOURCE_STATUS_MSIS
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_PLL1

Reference Manual to LL API cross reference:

- CFGR1 SWS LL_RCC_GetSysClkSource

LL_RCC_SetAHBPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetAHBPrescaler (uint32_t Prescaler)

Function description

Set AHB prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 HPRE LL_RCC_SetAHBPrescaler

LL_RCC_SetSystickClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSystickClockSource (uint32_t SystickSource)

Function description

Set Systick clock source.

Parameters

- **SystickSource:** This parameter can be one of the following values:
  – LL_RCC_SYSTICK_CLKSOURCE_LSI
  – LL_RCC_SYSTICK_CLKSOURCE_LSE
  – LL_RCC_SYSTICK_CLKSOURCE_HCLKDIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 SYSTICKSEL LL_RCC_SetSystickClockSource

LL_RCC_SetAPB1Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB1Prescaler (uint32_t Prescaler)

Function description

Set APB1 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  – LL_RCC_APB1_DIV_1
  – LL_RCC_APB1_DIV_2
  – LL_RCC_APB1_DIV_4
  – LL_RCC_APB1_DIV_8
  – LL_RCC_APB1_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE1 LL_RCC_SetAPB1Prescaler

LL_RCC_SetAPB2Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB2Prescaler (uint32_t Prescaler)

Function description

Set APB2 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE2 LL_RCC_SetAPB2Prescaler

LL_RCC_SetAPB3Prescaler

Function name

__STATIC_INLINE void LL_RCC_SetAPB3Prescaler (uint32_t Prescaler)

Function description

Set APB3 prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR3 PPRE3 LL_RCC_SetAPB3Prescaler

LL_RCC_SetDPHYPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetDPHYPrescaler (uint32_t Prescaler)

Function description

Set DPHY clock prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_DPHY_DIV_1
  - LL_RCC_DPHY_DIV_2
  - LL_RCC_DPHY_DIV_4
  - LL_RCC_DPHY_DIV_8
  - LL_RCC_DPHY_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR2 PPRE_DPHY LL_RCC_SetDPHYPrescaler

LL_RCC_GetAHBPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAHBPrescaler (void)

Function description

Get AHB prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_8
  - LL_RCC_SYSCLK_DIV_16
  - LL_RCC_SYSCLK_DIV_64
  - LL_RCC_SYSCLK_DIV_128
  - LL_RCC_SYSCLK_DIV_256
  - LL_RCC_SYSCLK_DIV_512

Reference Manual to LL API cross reference:

- CFGR2 HPRE LL_RCC_GetAHBPrescaler

LL_RCC_GetSystickClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSystickClockSource (void)

Function description

Get Sysctick clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SYSTICK_CLKSOURCE_LSI
  - LL_RCC_SYSTICK_CLKSOURCE_LSE
  - LL_RCC_SYSTICK_CLKSOURCE_HCLKDIV8

Reference Manual to LL API cross reference:

- CCIPR1 SYSTICKSEL LL_RCC_SetSystickClockSource

LL_RCC_GetAPB1Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB1Prescaler (void)

Function description

Get APB1 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB1_DIV_1
  - LL_RCC_APB1_DIV_2
  - LL_RCC_APB1_DIV_4
  - LL_RCC_APB1_DIV_8
  - LL_RCC_APB1_DIV_16

Reference Manual to LL API cross reference:

- CFGR2 PPRE1 LL_RCC_GetAPB1Prescaler

LL_RCC_GetAPB2Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB2Prescaler (void)

Function description

Get APB2 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB2_DIV_1
  - LL_RCC_APB2_DIV_2
  - LL_RCC_APB2_DIV_4
  - LL_RCC_APB2_DIV_8
  - LL_RCC_APB2_DIV_16

Reference Manual to LL API cross reference:

- CFGR2 PPRE2 LL_RCC_GetAPB2Prescaler

LL_RCC_GetAPB3Prescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetAPB3Prescaler (void)

Function description

Get APB3 prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_APB3_DIV_1
  - LL_RCC_APB3_DIV_2
  - LL_RCC_APB3_DIV_4
  - LL_RCC_APB3_DIV_8
  - LL_RCC_APB3_DIV_16

Reference Manual to LL API cross reference:

- CFGR3 PPRE3 LL_RCC_GetAPB2Prescaler

LL_RCC_GetDPHYPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDPHYPrescaler (void)

Function description

Get DPHY clock prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_DPHY_DIV_1
  - LL_RCC_DPHY_DIV_2
  - LL_RCC_DPHY_DIV_4
  - LL_RCC_DPHY_DIV_8
  - LL_RCC_DPHY_DIV_16

Reference Manual to LL API cross reference:

- CFGR2 PPRE_DPHY LL_RCC_GetDPHYPrescaler

LL_RCC_SetClkAfterWakeFromStop

Function name

__STATIC_INLINE void LL_RCC_SetClkAfterWakeFromStop (uint32_t Clock)

Function description

Set Clock After Wake-Up From Stop mode.

Parameters

- **Clock:** This parameter can be one of the following values:
  - LL_RCC_STOP_WAKEUPCLOCK_MSIS
  - LL_RCC_STOP_WAKEUPCLOCK_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 STOPWUCK LL_RCC_SetClkAfterWakeFromStop

LL_RCC_GetClkAfterWakeFromStop

Function name

__STATIC_INLINE uint32_t LL_RCC_GetClkAfterWakeFromStop (void)

Function description

Get Clock After Wake-Up From Stop mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_STOP_WAKEUPCLOCK_MSIS
  - LL_RCC_STOP_WAKEUPCLOCK_HSI

Reference Manual to LL API cross reference:

- CFGR1 STOPWUCK LL_RCC_GetClkAfterWakeFromStop

LL_RCC_SetKerClkAfterWakeFromStop

Function name

__STATIC_INLINE void LL_RCC_SetKerClkAfterWakeFromStop (uint32_t Clock)

Function description

Set Kernel Clock After Wake-Up From Stop mode.

Parameters

- **Clock:** This parameter can be one of the following values:
  – LL_RCC_STOP_WAKEUPKERCLOCK_MSIK
  – LL_RCC_STOP_WAKEUPKERCLOCK_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 STOPKERWUCK LL_RCC_SetKerClkAfterWakeFromStop

LL_RCC_GetKerClkAfterWakeFromStop

Function name

__STATIC_INLINE uint32_t LL_RCC_GetKerClkAfterWakeFromStop (void)

Function description

Get Kernel Clock After Wake-Up From Stop mode.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_STOP_WAKEUPKERCLOCK_MSIK
  – LL_RCC_STOP_WAKEUPKERCLOCK_HSI

Reference Manual to LL API cross reference:

- CFGR1 STOPKERWUCK LL_RCC_GetKerClkAfterWakeFromStop

LL_RCC_ConfigMCO

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOxSource, uint32_t MCOxPrescaler)

Function description

Configure MCOx.

Parameters

- **MCOxSource:** This parameter can be one of the following values:
  – LL_RCC_MCO1SOURCE_NOCLOCK
  – LL_RCC_MCO1SOURCE_SYSCLK
  – LL_RCC_MCO1SOURCE_MSIS
  – LL_RCC_MCO1SOURCE_HSI
  – LL_RCC_MCO1SOURCE_HSE
  – LL_RCC_MCO1SOURCE_HSI48
  – LL_RCC_MCO1SOURCE_PLLCLK
  – LL_RCC_MCO1SOURCE_LSI
  – LL_RCC_MCO1SOURCE_LSE
- **MCOxPrescaler:** This parameter can be one of the following values:
  – LL_RCC_MCO1_DIV_1
  – LL_RCC_MCO1_DIV_2
  – LL_RCC_MCO1_DIV_4
  – LL_RCC_MCO1_DIV_8
  – LL_RCC_MCO1_DIV_16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR1 MCOSEL LL_RCC_ConfigMCO
- CFGR1 MCOPRE LL_RCC_ConfigMCO

LL_RCC_SetUSARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSARTClockSource (uint32_t USARTxSource)

Function description

Configure USARTx clock source.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  – LL_RCC_USART1_CLKSOURCE_PCLK2
  – LL_RCC_USART1_CLKSOURCE_SYSCLK
  – LL_RCC_USART1_CLKSOURCE_HSI
  – LL_RCC_USART1_CLKSOURCE_LSE
  – LL_RCC_USART2_CLKSOURCE_PCLK1 (*)
  – LL_RCC_USART2_CLKSOURCE_SYSCLK (*)
  – LL_RCC_USART2_CLKSOURCE_HSI (*)
  – LL_RCC_USART2_CLKSOURCE_LSE (*)
  – LL_RCC_USART3_CLKSOURCE_PCLK1
  – LL_RCC_USART3_CLKSOURCE_SYSCLK
  – LL_RCC_USART3_CLKSOURCE_HSI
  – LL_RCC_USART3_CLKSOURCE_LSE
  – LL_RCC_USART6_CLKSOURCE_PCLK1 (*)
  – LL_RCC_USART6_CLKSOURCE_SYSCLK (*)
  – LL_RCC_USART6_CLKSOURCE_HSI (*)
  – LL_RCC_USART6_CLKSOURCE_LSE (*)

  (*) Availability depends on devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 USART1SEL LL_RCC_SetUSARTClockSource
- CCIPR1 USART2SEL LL_RCC_SetUSARTClockSource
- CCIPR1 USART3SEL LL_RCC_SetUSARTClockSource
- CCIPR2 USART6SEL LL_RCC_SetUSARTClockSource

LL_RCC_SetUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUARTClockSource (uint32_t UARTxSource)

Function description

Configure UARTx clock source.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  – LL_RCC_UART4_CLKSOURCE_PCLK1
  – LL_RCC_UART4_CLKSOURCE_SYSCLK
  – LL_RCC_UART4_CLKSOURCE_HSI
  – LL_RCC_UART4_CLKSOURCE_LSE
  – LL_RCC_UART5_CLKSOURCE_PCLK1
  – LL_RCC_UART5_CLKSOURCE_SYSCLK
  – LL_RCC_UART5_CLKSOURCE_HSI
  – LL_RCC_UART5_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 UART4SEL LL_RCC_SetUARTClockSource
- CCIPR1 UART5SEL LL_RCC_SetUARTClockSource

LL_RCC_SetLPUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t LPUARTxSource)

Function description

Configure LPUARTx clock source.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  – LL_RCC_LPUART1_CLKSOURCE_PCLK3
  – LL_RCC_LPUART1_CLKSOURCE_SYSCLK
  – LL_RCC_LPUART1_CLKSOURCE_HSI
  – LL_RCC_LPUART1_CLKSOURCE_LSE
  – LL_RCC_LPUART1_CLKSOURCE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 LPUART1SEL LL_RCC_SetLPUARTClockSource

LL_RCC_SetI2CClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2CClockSource (uint32_t I2CxSource)

Function description

Configure I2Cx clock source.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_SYSCLK
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C1_CLKSOURCE_MSIK
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_SYSCLK
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_MSIK
  - LL_RCC_I2C3_CLKSOURCE_PCLK3
  - LL_RCC_I2C3_CLKSOURCE_SYSCLK
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_MSIK
  - LL_RCC_I2C4_CLKSOURCE_PCLK1
  - LL_RCC_I2C4_CLKSOURCE_SYSCLK
  - LL_RCC_I2C4_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 I2C1SEL LL_RCC_SetI2CClockSource
- CCIPR1 I2C2SEL LL_RCC_SetI2CClockSource
- CCIPR3 I2C3SEL LL_RCC_SetI2CClockSource
- CCIPR1 I2C4SEL LL_RCC_SetI2CClockSource

LL_RCC_SetSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPIClockSource (uint32_t SPIxSource)

Function description

Configure SPIx clock source.

Parameters

- **SPIxSource:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE_PCLK2
  - LL_RCC_SPI1_CLKSOURCE_SYSCLK
  - LL_RCC_SPI1_CLKSOURCE_HSI
  - LL_RCC_SPI1_CLKSOURCE_MSIK
  - LL_RCC_SPI2_CLKSOURCE_PCLK1
  - LL_RCC_SPI2_CLKSOURCE_SYSCLK
  - LL_RCC_SPI2_CLKSOURCE_HSI
  - LL_RCC_SPI2_CLKSOURCE_MSIK
  - LL_RCC_SPI3_CLKSOURCE_PCLK3
  - LL_RCC_SPI3_CLKSOURCE_SYSCLK
  - LL_RCC_SPI3_CLKSOURCE_HSI
  - LL_RCC_SPI3_CLKSOURCE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 SPI1SEL LL_RCC_SetSPIClockSource
- CCIPR1 SPI1SEL LL_RCC_SetSPIClockSource
- CCIPR1 SPI2SEL LL_RCC_SetSPIClockSource
- CCIPR3 SPI3SEL LL_RCC_SetSPIClockSource

LL_RCC_SetLPTIMClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPTIMClockSource (uint32_t LPTIMxSource)

Function description

Configure LPTIMx clock source.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  – LL_RCC_LPTIM1_CLKSOURCE_MSIK
  – LL_RCC_LPTIM1_CLKSOURCE_LSI
  – LL_RCC_LPTIM1_CLKSOURCE_HSI
  – LL_RCC_LPTIM1_CLKSOURCE_LSE
  – LL_RCC_LPTIM2_CLKSOURCE_PCLK1
  – LL_RCC_LPTIM2_CLKSOURCE_LSI
  – LL_RCC_LPTIM2_CLKSOURCE_HSI
  – LL_RCC_LPTIM2_CLKSOURCE_LSE
  – LL_RCC_LPTIM34_CLKSOURCE_MSIK
  – LL_RCC_LPTIM34_CLKSOURCE_LSI
  – LL_RCC_LPTIM34_CLKSOURCE_HSI
  – LL_RCC_LPTIM34_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 LPTIM2SEL LL_RCC_SetLPTIMClockSource
- CCIPR3 LPTIM2SEL LL_RCC_SetLPTIMClockSource
- CCIPR3 LPTIM34SEL LL_RCC_SetLPTIMClockSource

LL_RCC_SetFDCANClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFDCANClockSource (uint32_t FDCANxSource)

Function description

Configure FDCAN kernel clock source.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  – LL_RCC_FDCAN_CLKSOURCE_HSE
  – LL_RCC_FDCAN_CLKSOURCE_PLL1
  – LL_RCC_FDCAN_CLKSOURCE_PLL2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 FDCANSEL LL_RCC_SetFDCANClockSource

LL_RCC_SetSAIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSAIClockSource (uint32_t SAIxSource)

Function description

Configure SAIx clock source.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  – LL_RCC_SAI1_CLKSOURCE_PLL1
  – LL_RCC_SAI1_CLKSOURCE_PLL2
  – LL_RCC_SAI1_CLKSOURCE_PLL3
  – LL_RCC_SAI1_CLKSOURCE_PIN
  – LL_RCC_SAI1_CLKSOURCE_HSI
  – LL_RCC_SAI2_CLKSOURCE_PLL1 (*)
  – LL_RCC_SAI2_CLKSOURCE_PLL2 (*)
  – LL_RCC_SAI2_CLKSOURCE_PLL3 (*)
  – LL_RCC_SAI2_CLKSOURCE_PIN (*)
  – LL_RCC_SAI2_CLKSOURCE_HSI (*)

  (*) Availability depends on devices.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 SAI1SEL LL_RCC_SetSAIClockSource
- CCIPR2 SAI2SEL LL_RCC_SetSAIClockSource

LL_RCC_SetSDMMCKernelClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSDMMCKernelClockSource (uint32_t SDMMCxSource)

Function description

Configure SDMMC1/2 kernel clock source.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  – LL_RCC_SDMMC12_KERNELCLKSOURCE_48CLK
  – P

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 SDMMCSEL LL_RCC_SetSDMMCKernelClockSource

LL_RCC_SetSDMMCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSDMMCClockSource (uint32_t SDMMCxSource)

Function description

Configure SDMMC1/2 clock source.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  – LL_RCC_SDMMC12_CLKSOURCE_HSI48
  – LL_RCC_SDMMC12_CLKSOURCE_PLL1
  – LL_RCC_SDMMC12_CLKSOURCE_PLL2
  – LL_RCC_SDMMC12_CLKSOURCE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 ICLKSEL LL_RCC_SetSDMMCClockSource

LL_RCC_SetRNGClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRNGClockSource (uint32_t RNGxSource)

Function description

Configure RNG clock source.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  – LL_RCC_RNG_CLKSOURCE_HSI48
  – LL_RCC_RNG_CLKSOURCE_HSI48_DIV2
  – LL_RCC_RNG_CLKSOURCE_HSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 RNGSEL LL_RCC_SetRNGClockSource

LL_RCC_SetUSBPHYClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSBPHYClockSource (uint32_t Source)

Function description

Configure USBPHY clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_USBPHYCLKSOURCE_HSE
  – LL_RCC_USBPHYCLKSOURCE_HSE_DIV2
  – LL_RCC_USBPHYCLKSOURCE_PLL1
  – LL_RCC_USBPHYCLKSOURCE_PLL1_DIV2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 USBPHYCSEL LL_RCC_SetUSBPHYClockSource

LL_RCC_SetUSBClockSource

Function name

__STATIC_INLINE void LL_RCC_SetUSBClockSource (uint32_t USBxSource)

Function description

Configure USB clock source.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  - LL_RCC_USB_CLKSOURCE_HSI48
  - LL_RCC_USB_CLKSOURCE_PLL1
  - LL_RCC_USB_CLKSOURCE_PLL2
  - LL_RCC_USB_CLKSOURCE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR1 ICLKSEL LL_RCC_SetUSBClockSource

LL_RCC_SetADCDACClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADCDACClockSource (uint32_t ADCxDAC1Source)

Function description

Configure ADC clock source.

Parameters

- **ADCxDAC1Source:** This parameter can be one of the following values:
  - LL_RCC_ADCDAC_CLKSOURCE_HCLK
  - LL_RCC_ADCDAC_CLKSOURCE_SYSCLK
  - LL_RCC_ADCDAC_CLKSOURCE_PLL2
  - LL_RCC_ADCDAC_CLKSOURCE_HSE
  - LL_RCC_ADCDAC_CLKSOURCE_HSI
  - LL_RCC_ADCDAC_CLKSOURCE_MSIK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 ADCDACSEL LL_RCC_SetADCDACClockSource

LL_RCC_SetDAC1ClockSource

Function name

__STATIC_INLINE void LL_RCC_SetDAC1ClockSource (uint32_t Source)

Function description

Configure DAC1 clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_DAC1_CLKSOURCE_LSE
  - LL_RCC_DAC1_CLKSOURCE_LSI

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 DAC1SEL LL_RCC_SetDAC1ClockSource

LL_RCC_SetADF1ClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADF1ClockSource (uint32_t Source)

Function description

Configure ADF1 clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_ADF1_CLKSOURCE_HCLK
  - LL_RCC_ADF1_CLKSOURCE_PLL1
  - LL_RCC_ADF1_CLKSOURCE_PLL3
  - LL_RCC_ADF1_CLKSOURCE_MSIK
  - LL_RCC_ADF1_CLKSOURCE_PIN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 ADF1SEL LL_RCC_SetADF1ClockSource

LL_RCC_SetMDF1ClockSource

Function name

__STATIC_INLINE void LL_RCC_SetMDF1ClockSource (uint32_t Source)

Function description

Configure MDF1 clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_MDF1_CLKSOURCE_HCLK
  - LL_RCC_MDF1_CLKSOURCE_PLL1
  - LL_RCC_MDF1_CLKSOURCE_PLL3
  - LL_RCC_MDF1_CLKSOURCE_MSIK
  - LL_RCC_MDF1_CLKSOURCE_PIN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR3 MDF1SEL LL_RCC_SetMDF1ClockSource

LL_RCC_SetOCTOSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetOCTOSPIClockSource (uint32_t Source)

Function description

Configure OCTOSPI kernel clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_OCTOSPI_CLKSOURCE_SYSCLK
  - LL_RCC_OCTOSPI_CLKSOURCE_MSIK
  - LL_RCC_OCTOSPI_CLKSOURCE_PLL1
  - LL_RCC_OCTOSPI_CLKSOURCE_PLL2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 OSPISEL LL_RCC_SetOCTOSPIClockSource

LL_RCC_SetHSPIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetHSPIClockSource (uint32_t Source)

Function description

Configure HSPI kernel clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_HSPI_CLKSOURCE_SYSCLK
  - LL_RCC_HSPI_CLKSOURCE_PLL1
  - LL_RCC_HSPI_CLKSOURCE_PLL2
  - LL_RCC_HSPI_CLKSOURCE_PLL3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 HSPISEL LL_RCC_SetHSPIClockSource

LL_RCC_SetSAESClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSAESClockSource (uint32_t Source)

Function description

Configure SAES clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_SAES_CLKSOURCE_SHSI
  - LL_RCC_SAES_CLKSOURCE_SHSI_DIV2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 SAESSEL LL_RCC_SetSAESClockSource

LL_RCC_SetDSIClockSource

Function name

__STATIC_INLINE void LL_RCC_SetDSIClockSource (uint32_t Source)

Function description

Configure DSIx clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_DSI_CLKSOURCE_PHY
  – LL_RCC_DSI_CLKSOURCE_PLL3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 DSISEL LL_RCC_SetDSIClockSource

LL_RCC_SetLTDCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLTDCClockSource (uint32_t Source)

Function description

Configure LTDCx clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_LTDC_CLKSOURCE_PLL2
  – LL_RCC_LTDC_CLKSOURCE_PLL3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR2 LTDCSEL LL_RCC_SetLTDCClockSource

LL_RCC_GetUSARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSARTClockSource (uint32_t USARTx)

Function description

Get USARTx clock source.

Parameters

- **USARTx:** This parameter can be one of the following values:
  – LL_RCC_USART1_CLKSOURCE
  – LL_RCC_USART2_CLKSOURCE (*)
  – LL_RCC_USART3_CLKSOURCE
  – LL_RCC_USART6_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_USART1_CLKSOURCE_PCLK2
  – LL_RCC_USART1_CLKSOURCE_SYSCLK
  – LL_RCC_USART1_CLKSOURCE_HSI
  – LL_RCC_USART1_CLKSOURCE_LSE
  – LL_RCC_USART2_CLKSOURCE_PCLK1 (*)
  – LL_RCC_USART2_CLKSOURCE_SYSCLK (*)
  – LL_RCC_USART2_CLKSOURCE_HSI (*)
  – LL_RCC_USART2_CLKSOURCE_LSE (*)
  – LL_RCC_USART3_CLKSOURCE_PCLK1
  – LL_RCC_USART3_CLKSOURCE_SYSCLK
  – LL_RCC_USART3_CLKSOURCE_HSI
  – LL_RCC_USART3_CLKSOURCE_LSE
  – LL_RCC_USART6_CLKSOURCE_PCLK1 (*)
  – LL_RCC_USART6_CLKSOURCE_SYSCLK (*)
  – LL_RCC_USART6_CLKSOURCE_HSI (*)
  – LL_RCC_USART6_CLKSOURCE_LSE (*)

  (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CCIPR1 USART1SEL LL_RCC_GetUSARTClockSource
- CCIPR1 USART2SEL LL_RCC_GetUSARTClockSource
- CCIPR1 USART3SEL LL_RCC_GetUSARTClockSource
- CCIPR2 USART6SEL LL_RCC_GetUSARTClockSource

LL_RCC_GetUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUARTClockSource (uint32_t UARTx)

Function description

Get UARTx clock source.

Parameters

- **UARTx:** This parameter can be one of the following values:
  – LL_RCC_UART4_CLKSOURCE
  – LL_RCC_UART5_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_UART4_CLKSOURCE_PCLK1
  – LL_RCC_UART4_CLKSOURCE_SYSCLK
  – LL_RCC_UART4_CLKSOURCE_HSI
  – LL_RCC_UART4_CLKSOURCE_LSE
  – LL_RCC_UART5_CLKSOURCE_PCLK1
  – LL_RCC_UART5_CLKSOURCE_SYSCLK
  – LL_RCC_UART5_CLKSOURCE_HSI
  – LL_RCC_UART5_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR1 UART4SEL LL_RCC_GetUARTClockSource
- CCIPR1 UART5SEL LL_RCC_GetUARTClockSource

LL_RCC_GetLPUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (uint32_t LPUARTx)

Function description

Get LPUARTx clock source.

Parameters

- **LPUARTx:** This parameter can be one of the following values:
  – LL_RCC_LPUART1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_LPUART1_CLKSOURCE_PCLK3
  – LL_RCC_LPUART1_CLKSOURCE_SYSCLK
  – LL_RCC_LPUART1_CLKSOURCE_HSI
  – LL_RCC_LPUART1_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR1 LPUART1SEL LL_RCC_GetLPUARTClockSource

LL_RCC_GetI2CClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2CClockSource (uint32_t I2Cx)

Function description

Get I2Cx clock source.

Parameters

- **I2Cx:** This parameter can be one of the following values:
  – LL_RCC_I2C1_CLKSOURCE
  – LL_RCC_I2C2_CLKSOURCE
  – LL_RCC_I2C3_CLKSOURCE
  – LL_RCC_I2C4_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE_PCLK1
  - LL_RCC_I2C1_CLKSOURCE_SYSCLK
  - LL_RCC_I2C1_CLKSOURCE_HSI
  - LL_RCC_I2C2_CLKSOURCE_PCLK1
  - LL_RCC_I2C2_CLKSOURCE_SYSCLK
  - LL_RCC_I2C2_CLKSOURCE_HSI
  - LL_RCC_I2C3_CLKSOURCE_PCLK3
  - LL_RCC_I2C3_CLKSOURCE_SYSCLK
  - LL_RCC_I2C3_CLKSOURCE_HSI
  - LL_RCC_I2C4_CLKSOURCE_PCLK1
  - LL_RCC_I2C4_CLKSOURCE_SYSCLK
  - LL_RCC_I2C4_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR1 I2C1SEL LL_RCC_GetI2CClockSource
- CCIPR1 I2C2SEL LL_RCC_GetI2CClockSource
- CCIPR3 I2C3SEL LL_RCC_GetI2CClockSource
- CCIPR1 I2C4SEL LL_RCC_GetI2CClockSource

LL_RCC_GetSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPIClockSource (uint32_t SPIx)

Function description

Get SPIx clock source.

Parameters

- **SPIx:** This parameter can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE
  - LL_RCC_SPI2_CLKSOURCE
  - LL_RCC_SPI3_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SPI1_CLKSOURCE_PCLK2
  - LL_RCC_SPI1_CLKSOURCE_SYSCLK
  - LL_RCC_SPI1_CLKSOURCE_HSI
  - LL_RCC_SPI1_CLKSOURCE_MSIK
  - LL_RCC_SPI2_CLKSOURCE_PCLK1
  - LL_RCC_SPI2_CLKSOURCE_SYSCLK
  - LL_RCC_SPI2_CLKSOURCE_HSI
  - LL_RCC_SPI2_CLKSOURCE_MSIK
  - LL_RCC_SPI3_CLKSOURCE_PCLK3
  - LL_RCC_SPI3_CLKSOURCE_SYSCLK
  - LL_RCC_SPI3_CLKSOURCE_HSI
  - LL_RCC_SPI3_CLKSOURCE_MSIK

Reference Manual to LL API cross reference:

- CCIPR1 SPI1SEL LL_RCC_GetSPIClockSource
- CCIPR1 SPI2SEL LL_RCC_GetSPIClockSource
- CCIPR3 SPI3SEL LL_RCC_GetSPIClockSource

LL_RCC_GetLPTIMClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPTIMClockSource (uint32_t LPTIMx)

Function description

Get LPTIMx clock source.

Parameters

- **LPTIMx:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM34_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE_MSIK
  - LL_RCC_LPTIM1_CLKSOURCE_LSI
  - LL_RCC_LPTIM1_CLKSOURCE_HSI
  - LL_RCC_LPTIM1_CLKSOURCE_LSE
  - LL_RCC_LPTIM2_CLKSOURCE_PCLK1
  - LL_RCC_LPTIM2_CLKSOURCE_LSI
  - LL_RCC_LPTIM2_CLKSOURCE_HSI
  - LL_RCC_LPTIM2_CLKSOURCE_LSE
  - LL_RCC_LPTIM34_CLKSOURCE_MSIK
  - LL_RCC_LPTIM34_CLKSOURCE_LSI
  - LL_RCC_LPTIM34_CLKSOURCE_HSI
  - LL_RCC_LPTIM34_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR1 LPTIM2SEL LL_RCC_GetLPTIMClockSource
- CCIPR3 LPTIM2SEL LL_RCC_GetLPTIMClockSource
- CCIPR3 LPTIM34SEL LL_RCC_GetLPTIMClockSource

LL_RCC_SetTIMICClockSource

Function name

__STATIC_INLINE void LL_RCC_SetTIMICClockSource (uint32_t TIMICSource)

Function description

Set Tim Input capture clock source.

Parameters

- **TIMICSource:** This parameter can be one of the following combined values:
  - LL_RCC_TIMIC_CLKSOURCE_NONE
  - LL_RCC_TIMIC_CLKSOURCE_HSI_DIV256
  - LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV1024
  - LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV4
  - LL_RCC_TIMIC_CLKSOURCE_MSIK_DIV4
  - LL_RCC_TIMIC_CLKSOURCE_MSIK_DIV1024

Return values

- **None:**

Notes

- HSI, MSI and MSIK clocks without division are also available when TIMICSEL[2] is 1.
- combination to be avoided : LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV1024 and LL_RCC_CLKSOURCE_TIMIC_MSIK_DIV1024 LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV4 and LL_RCC_CLKSOURCE_TIMIC_MSIK_DIV4

Reference Manual to LL API cross reference:

- CCIPR1 TIMICSEL LL_RCC_SetTIMICClockSource

LL_RCC_GetTIMICClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetTIMICClockSource (void)

Function description

Get Tim Input capture clock source.

Return values

- **Returned:** value can be one of the following combined values:
  - LL_RCC_TIMIC_CLKSOURCE_NONE
  - LL_RCC_TIMIC_CLKSOURCE_HSI_DIV256
  - LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV1024
  - LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV4
  - LL_RCC_TIMIC_CLKSOURCE_MSIK_DIV4
  - LL_RCC_TIMIC_CLKSOURCE_MSIK_DIV1024

Reference Manual to LL API cross reference:

- CCIPR1 TIMICSEL LL_RCC_GetTIMICClockSource

LL_RCC_GetFDCANClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFDCANClockSource (uint32_t FDCANx)

Function description

Get FDCAN kernel clock source.

Parameters

- **FDCANx:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE_HSE
  - LL_RCC_FDCAN_CLKSOURCE_PLL1
  - LL_RCC_FDCAN_CLKSOURCE_PLL2

Reference Manual to LL API cross reference:

- CCIPR1 FDCANSEL LL_RCC_GetFDCANClockSource

LL_RCC_GetSAIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSAIClockSource (uint32_t SAIx)

Function description

Get SAIx clock source.

Parameters

- **SAIx:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE_PLL1
  - LL_RCC_SAI1_CLKSOURCE_PLL2
  - LL_RCC_SAI1_CLKSOURCE_PLL3
  - LL_RCC_SAI1_CLKSOURCE_PIN
  - LL_RCC_SAI1_CLKSOURCE_HSI
  - LL_RCC_SAI2_CLKSOURCE_PLL1 (*)
  - LL_RCC_SAI2_CLKSOURCE_PLL2 (*)
  - LL_RCC_SAI2_CLKSOURCE_PLL3 (*)
  - LL_RCC_SAI2_CLKSOURCE_PIN (*)
  - LL_RCC_SAI2_CLKSOURCE_HSI (*)

  (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- CCIPR2 SAI1SEL LL_RCC_GetSAIClockSource
- CCIPR2 SAI2SEL LL_RCC_GetSAIClockSource

LL_RCC_GetSDMMCKernelClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSDMMCKernelClockSource (uint32_t SDMMCx)

Function description

Get SDMMCx kernel clock source.

Parameters

- **SDMMCx:** This parameter can be one of the following values:
  - LL_RCC_SDMMC_KERNELCLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SDMMC12_KERNELCLKSOURCE_48CLK
  - P

Reference Manual to LL API cross reference:

- CCIPR2 SDMMCSEL LL_RCC_GetSDMMCKernelClockSource

LL_RCC_GetSDMMCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSDMMCClockSource (uint32_t SDMMCx)

Function description

Get SDMMC1/2 clock source.

Parameters

- **SDMMCx:** This parameter can be one of the following values:
  - LL_RCC_SDMMC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_SDMMC12_CLKSOURCE_HSI48
  - LL_RCC_SDMMC12_CLKSOURCE_PLL1
  - LL_RCC_SDMMC12_CLKSOURCE_PLL2
  - LL_RCC_SDMMC12_CLKSOURCE_MSIK

Reference Manual to LL API cross reference:

- CCIPR1 ICLKSEL LL_RCC_GetSDMMCClockSource

LL_RCC_GetRNGClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRNGClockSource (uint32_t RNGx)

Function description

Get RNGx clock source.

Parameters

- **RNGx:** This parameter can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RNG_CLKSOURCE_HSI48
  - LL_RCC_RNG_CLKSOURCE_HSI48_DIV2
  - LL_RCC_RNG_CLKSOURCE_HSI

Reference Manual to LL API cross reference:

- CCIPR2 RNGSEL LL_RCC_GetRNGClockSource

LL_RCC_GetUSBPHYClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSBPHYClockSource (uint32_t USBPHYx)

Function description

Get USBPHYx clock source.

Parameters

- **USBPHYx:** This parameter can be one of the following values:
  – LL_RCC_USBPHY_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_USBPHYCLKSOURCE_HSE
  – LL_RCC_USBPHYCLKSOURCE_HSE_DIV2
  – LL_RCC_USBPHYCLKSOURCE_PLL1
  – LL_RCC_USBPHYCLKSOURCE_PLL1_DIV2

Reference Manual to LL API cross reference:

- CCIPR2 USBPHYCSEL LL_RCC_GetUSBPHYClockSource

LL_RCC_GetUSBClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetUSBClockSource (uint32_t USBx)

Function description

Get USBx clock source.

Parameters

- **USBx:** This parameter can be one of the following values:
  – LL_RCC_USB_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_USB_CLKSOURCE_HSI48
  – LL_RCC_USB_CLKSOURCE_PLL1
  – LL_RCC_USB_CLKSOURCE_PLL2
  – LL_RCC_USB_CLKSOURCE_MSIK

Reference Manual to LL API cross reference:

- CCIPR1 ICLKSEL LL_RCC_GetUSBClockSource

LL_RCC_GetADCDACClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCDACClockSource (uint32_t ADCxDAC1)

Function description

Get ADCx clock source.

Parameters

- **ADCxDAC1:** This parameter can be one of the following values:
  – LL_RCC_ADCDAC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_ADCDAC_CLKSOURCE_HCLK
  – LL_RCC_ADCDAC_CLKSOURCE_SYSCLK
  – LL_RCC_ADCDAC_CLKSOURCE_PLL2
  – LL_RCC_ADCDAC_CLKSOURCE_MSIK
  – LL_RCC_ADCDAC_CLKSOURCE_HSE

Reference Manual to LL API cross reference:

- CCIPR3 ADCDACSEL LL_RCC_SetADCDACClockSource

LL_RCC_GetADF1ClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADF1ClockSource (uint32_t ADF1x)

Function description

Get DFSDM Audio Clock Source.

Parameters

- **ADF1x:** This parameter can be one of the following values:
  – LL_RCC_ADF1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_ADF1_CLKSOURCE_HCLK
  – LL_RCC_ADF1_CLKSOURCE_PLL1
  – LL_RCC_ADF1_CLKSOURCE_PLL3
  – LL_RCC_ADF1_CLKSOURCE_MSIK
  – LL_RCC_ADF1_CLKSOURCE_PIN

Reference Manual to LL API cross reference:

- CCIPR3 ADF1SEL LL_RCC_GetADF1ClockSource

LL_RCC_GetDAC1ClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDAC1ClockSource (uint32_t DAC1x)

Function description

Get DAC1 Clock Source.

Parameters

- **DAC1x:** This parameter can be one of the following values:
  – LL_RCC_DAC1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_DAC1_CLKSOURCE_LSE
  – LL_RCC_DAC1_CLKSOURCE_LSI

Reference Manual to LL API cross reference:

- CCIPR3 DAC1SEL LL_RCC_GetDAC1ClockSource

LL_RCC_GetMDF1ClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetMDF1ClockSource (uint32_t MDF1x)

Function description

Get MDF1 Clock Source.

Parameters

- **MDF1x:** This parameter can be one of the following values:
  – LL_RCC_MDF1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_MDF1_CLKSOURCE_HCLK
  – LL_RCC_MDF1_CLKSOURCE_PLL1
  – LL_RCC_MDF1_CLKSOURCE_PLL3
  – LL_RCC_MDF1_CLKSOURCE_MSIK
  – LL_RCC_MDF1_CLKSOURCE_PIN

Reference Manual to LL API cross reference:

- CCIPR2 MDF1SEL LL_RCC_GetMDF1ClockSource

LL_RCC_GetOCTOSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetOCTOSPIClockSource (uint32_t OCTOSPIx)

Function description

Get OCTOSPI clock source.

Parameters

- **OCTOSPIx:** This parameter can be one of the following values:
  – LL_RCC_OCTOSPI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_OCTOSPI_CLKSOURCE_SYSCLK
  – LL_RCC_OCTOSPI_CLKSOURCE_MSIK
  – LL_RCC_OCTOSPI_CLKSOURCE_PLL1
  – LL_RCC_OCTOSPI_CLKSOURCE_PLL2

Reference Manual to LL API cross reference:

- CCIPR2 OSPISEL LL_RCC_GetOCTOSPIClockSource

LL_RCC_GetHSPIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetHSPIClockSource (uint32_t HSPIx)

Function description

Get HSPI clock source.

Parameters

- **HSPIx:** This parameter can be one of the following values:
  – LL_RCC_HSPI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_HSPI_CLKSOURCE_SYSCLK
  – LL_RCC_HSPI_CLKSOURCE_PLL1
  – LL_RCC_HSPI_CLKSOURCE_PLL2
  – LL_RCC_HSPI_CLKSOURCE_PLL3

Reference Manual to LL API cross reference:

- CCIPR2 HSPISEL LL_RCC_GetHSPIClockSource

LL_RCC_GetSAESClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSAESClockSource (uint32_t SAESx)

Function description

Get SAES kernel clock source.

Parameters

- **SAESx:** This parameter can be one of the following values:
  – LL_RCC_SAES_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_SAES_CLKSOURCE_SHSI
  – LL_RCC_SAES_CLKSOURCE_SHSI_DIV2

Reference Manual to LL API cross reference:

- CCIPR2 SAESSEL LL_RCC_GetSAESClockSource

LL_RCC_GetDSIClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDSIClockSource (uint32_t DSIx)

Function description

Get DSI clock source.

Parameters

- **DSIx:** This parameter can be one of the following values:
  – LL_RCC_DSI_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_DSI_CLKSOURCE_PHY
  – LL_RCC_DSI_CLKSOURCE_PLL3

Reference Manual to LL API cross reference:

- CCIPR2 DSISEL LL_RCC_GetDSIClockSource

LL_RCC_GetLTDCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLTDCClockSource (uint32_t LTDCx)

Function description

Get LTDC clock source.

Parameters

- **LTDCx:** This parameter can be one of the following values:
  – LL_RCC_LTDC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_LTDC_CLKSOURCE_PLL2
  – LL_RCC_LTDC_CLKSOURCE_PLL3

Reference Manual to LL API cross reference:

- CCIPR2 LTDCSEL LL_RCC_GetLTDCClockSource

LL_RCC_SetRTCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetRTCClockSource (uint32_t Source)

Function description

Set RTC Clock Source.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_RTC_CLKSOURCE_NONE
  – LL_RCC_RTC_CLKSOURCE_LSE
  – LL_RCC_RTC_CLKSOURCE_LSI
  – LL_RCC_RTC_CLKSOURCE_HSE_DIV32

Return values

- **None:**

Notes

- Once the RTC clock source has been selected, it cannot be changed anymore unless the Backup domain is reset, or unless a failure is detected on LSE (LSECSSD is set). The BDRST bit can be used to reset them.

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_SetRTCClockSource

LL_RCC_GetRTCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRTCClockSource (void)

Function description

Get RTC Clock Source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_RTC_CLKSOURCE_NONE
  - LL_RCC_RTC_CLKSOURCE_LSE
  - LL_RCC_RTC_CLKSOURCE_LSI
  - LL_RCC_RTC_CLKSOURCE_HSE_DIV32

Reference Manual to LL API cross reference:

- BDCR RTCSEL LL_RCC_GetRTCClockSource

LL_RCC_EnableRTC

Function name

__STATIC_INLINE void LL_RCC_EnableRTC (void)

Function description

Enable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_EnableRTC

LL_RCC_DisableRTC

Function name

__STATIC_INLINE void LL_RCC_DisableRTC (void)

Function description

Disable RTC.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_DisableRTC

LL_RCC_IsEnabledRTC

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledRTC (void)

Function description

Check if RTC has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- BDCR RTCEN LL_RCC_IsEnabledRTC

LL_RCC_ForceBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ForceBackupDomainReset (void)

Function description

Force the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST LL_RCC_ForceBackupDomainReset

LL_RCC_ReleaseBackupDomainReset

Function name

__STATIC_INLINE void LL_RCC_ReleaseBackupDomainReset (void)

Function description

Release the Backup domain reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- BDCR BDRST LL_RCC_ReleaseBackupDomainReset

LL_RCC_PLL1_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Enable (void)

Function description

Enable PLL1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL1ON LL_RCC_PLL1_Enable

LL_RCC_PLL1_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1_Disable (void)

Function description

Disable PLL1.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock

Reference Manual to LL API cross reference:

- CR PLLON LL_RCC_PLL1_Disable

LL_RCC_PLL1_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsReady (void)

Function description

Check if PLL1 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL1RDY LL_RCC_PLL1_IsReady

LL_RCC_PLL1_ConfigDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL1_ConfigDomain_SYS (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)

Function description

Configure PLL1 used for SYSCLK Domain.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLL1SOURCE_NONE
  - LL_RCC_PLL1SOURCE_MSIS
  - LL_RCC_PLL1SOURCE_HSI
  - LL_RCC_PLL1SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLR:** parameter can be a value between 1 and 128 (Only division by 1 and even division are allowed)
- **PLLN:** parameter can be a value between 4 and 512

Return values

- **None:**

Notes

- PLL1 Source, PLLM, PLLN and PLLR can be written only when PLL1 is disabled.
- PLLN/PLLR can be written only when PLL is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_ConfigDomain_SYS
- PLL1CFGR PLL1M LL_RCC_PLL1_ConfigDomain_SYS
- PLL1CFGR PLL1N LL_RCC_PLL1_ConfigDomain_SYS
- PLL1CFGR PLL1R LL_RCC_PLL1_ConfigDomain_SYS

LL_RCC_PLL1_ConfigDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL1_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)

Function description

Configure PLL1 used for SAI domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_PLL1SOURCE_NONE
  – LL_RCC_PLL1SOURCE_MSIS
  – LL_RCC_PLL1SOURCE_HSI
  – LL_RCC_PLL1SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLP:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL1 Source, PLLM, PLLN and PLLPDIV can be written only when PLL1 is disabled.
- This can be selected for SAI1 or SAI2

Reference Manual to LL API cross reference:

- PLLC1FGR PLL1SRC LL_RCC_PLL1_ConfigDomain_SAI
- PLLC1FGR PLL1M LL_RCC_PLL1_ConfigDomain_SAI
- PLLC1FGR PLL1N LL_RCC_PLL1_ConfigDomain_SAI
- PLLC1FGR PLL1P LL_RCC_PLL1_ConfigDomain_SAI

LL_RCC_PLL1_ConfigDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL1_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)

Function description

Configure PLL1 used for 48Mhz domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_PLL1SOURCE_NONE
  – LL_RCC_PLL1SOURCE_MSIS
  – LL_RCC_PLL1SOURCE_HSI
  – LL_RCC_PLL1SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLQ:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL1 Source, PLLM, PLLN and PLLQ can be written only when PLL1 is disabled.
- This can be selected for USB, SDMMC

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_ConfigDomain_48M
- PLL1CFGR PLL1M LL_RCC_PLL1_ConfigDomain_48M
- PLL1CFGR PLL1N LL_RCC_PLL1_ConfigDomain_48M
- PLL1CFGR PLL1Q LL_RCC_PLL1_ConfigDomain_48M

LL_RCC_PLL1_SetMainSource

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetMainSource (uint32_t PLL1Source)

Function description

Configure PLL clock source.

Parameters

- **PLL1Source:** This parameter can be one of the following values:
  – LL_RCC_PLL1SOURCE_NONE
  – LL_RCC_PLL1SOURCE_MSIS
  – LL_RCC_PLL1SOURCE_HSI
  – LL_RCC_PLL1SOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_SetMainSource

LL_RCC_PLL1_GetMainSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetMainSource (void)

Function description

Get the oscillator used as PLL1 clock source.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_PLL1SOURCE_NONE
  – LL_RCC_PLL1SOURCE_MSIS
  – LL_RCC_PLL1SOURCE_HSI
  – LL_RCC_PLL1SOURCE_HSE

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1SRC LL_RCC_PLL1_GetMainSource

LL_RCC_PLL1_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetN (uint32_t PLL1N)

Function description

Set Main PLL1 multiplication factor for VCO.

Parameters

- **PLL1N:** parameter can be a value between 4 and 512

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1N LL_RCC_PLL1_SetN

LL_RCC_PLL1_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetN (void)

Function description

Get Main PLL1 multiplication factor for VCO.

Return values

- **Between:** 4 and 512

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1N LL_RCC_PLL1_GetN

LL_RCC_PLL1_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetP (uint32_t PLL1P)

Function description

Set Main PLL1 division factor for PLL1P.

Parameters

- **PLL1P:** parameter can be a value between 1 and 128

Notes

- Used for SAI1 and SAI2 clock

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1P LL_RCC_PLL1_SetP

LL_RCC_PLL1_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetP (void)

Function description

Get Main PLL1 division factor for PLL1P.

Return values

- **Between:** 1 and 128

Notes

- Used for SAI1 and SAI2 clock

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1P LL_RCC_PLL1_GetP

LL_RCC_PLL1_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetQ (uint32_t PLL1Q)

Function description

Set Main PLL division factor for PLLQ.

Parameters

- **PLL1Q:** parameter can be a value between 1 and 128

Notes

- Used for PLL48M1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLLCFGR PLL1Q LL_RCC_PLL1_SetQ

LL_RCC_PLL1_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetQ (void)

Function description

Get Main PLL division factor for PLLQ.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL48M1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1Q LL_RCC_PLL1_GetQ

LL_RCC_PLL1_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetR (uint32_t PLL1R)

Function description

Set Main PLL division factor for PLL1R.

Parameters

- **PLL1R:** parameter can be a value between 1 and 128 (Only division by 1 and even division are allowed)

Notes

- Used for PLL1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLL1DIVR PLL1R LL_RCC_PLL1_SetR

LL_RCC_PLL1_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetR (void)

Function description

Get Main PLL1 division factor for PLL1R.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL1CLK (system clock)

Reference Manual to LL API cross reference:

- PLL1DIVR PLL1R LL_RCC_PLL1_GetR

LL_RCC_PLL1_SetDivider

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetDivider (uint32_t PLL1M)

Function description

Set Division factor for the main PLL and other PLL.

Parameters

- **PLL1M:** parameter can be a value between 1 and 16

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1M LL_RCC_PLL1_SetDivider

LL_RCC_PLL1_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetDivider (void)

Function description

Get Division factor for the main PLL and other PLL.

Return values

- **Between:** 1 and 16

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1M LL_RCC_PLL1_GetDivider

LL_RCC_PLL1_EnableDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableDomain_SAI (void)

Function description

Enable PLL1 output mapped on SAI domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1PEN LL_RCC_PLL1_EnableDomain_SAI

LL_RCC_PLL1_DisableDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableDomain_SAI (void)

Function description

Disable PLL1 output mapped on SAI domain clock.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock
- In order to save power, when the PLL1CLK of the PLL1 is not used, should be 0

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1PEN LL_RCC_PLL1_DisableDomain_SAI

LL_RCC_PLL1_IsEnabledDomain_SAI

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledDomain_SAI (void)

Function description

Check if PLL1 output mapped on SAI domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1PEN LL_RCC_PLL1_IsEnabledDomain_SAI

LL_RCC_PLL1_EnableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableDomain_48M (void)

Function description

Enable PLL output mapped on 48MHz domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1QEN LL_RCC_PLL1_EnableDomain_48M

LL_RCC_PLL1_DisableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableDomain_48M (void)

Function description

Disable PLL1 output mapped on 48MHz domain clock.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL clock is used as the system clock
- In order to save power, when the PLL1CLK of the PLL1 is not used, should be 0

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1QEN LL_RCC_PLL1_DisableDomain_48M

LL_RCC_PLL1_IsEnabledDomain_48M

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledDomain_48M (void)

Function description

Check if PLL1 output mapped on 48M domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1QEN LL_RCC_PLL1_IsEnabledDomain_48M

LL_RCC_PLL1_EnableDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL1_EnableDomain_SYS (void)

Function description

Enable PLL1 output mapped on SYSCLK domain.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1REN LL_RCC_PLL1_EnableDomain_SYS

LL_RCC_PLL1_DisableDomain_SYS

Function name

__STATIC_INLINE void LL_RCC_PLL1_DisableDomain_SYS (void)

Function description

Disable PLL1 output mapped on SYSCLK domain.

Return values

- **None:**

Notes

- Cannot be disabled if the PLL1 clock is used as the system clock
- In order to save power, when the PLL1CLK of the PLL1 is not used, Main PLL1 should be 0

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1REN LL_RCC_PLL1_DisableDomain_SYS

LL_RCC_PLL1_IsEnabledDomain_SYS

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_IsEnabledDomain_SYS (void)

Function description

Check if PLL1 output mapped on SYS domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1REN LL_RCC_PLL1_IsEnabledDomain_SYS

LL_RCC_PLL1FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL1FRACN_Enable (void)

Function description

Enable PLL1 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1FRACEN LL_RCC_PLL1FRACN_Enable

LL_RCC_PLL1FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1FRACN_IsEnabled (void)

Function description

Check if PLL1 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1FRACEN LL_RCC_PLL1FRACN_IsEnabled

LL_RCC_PLL1FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL1FRACN_Disable (void)

Function description

Disable PLL1 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1FRACEN LL_RCC_PLL1FRACN_Disable

LL_RCC_PLL1_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetFRACN (uint32_t FRACN)

Function description

Set PLL1 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL1FRACR PLL1FRACN LL_RCC_PLL1_SetFRACN

LL_RCC_PLL1_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL1_GetFRACN (void)

Function description

Get PLL1 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL1FRACR PLL1FRACN LL_RCC_PLL1_GetFRACN

LL_RCC_PLL1_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL1_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL1 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  – LL_RCC_PLLINPUTRANGE_4_8
  – LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL1 is disabled.

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1RGE LL_RCC_PLL1_SetVCOInputRange

LL_RCC_SetPll1EPodPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetPll1EPodPrescaler (uint32_t BoostDiv)

Function description

Set PLL1 EPOD Prescaler booster input clock.

Parameters

- **BoostDiv:** This parameter can be one of the following values:
  – LL_RCC_PLL1MBOOST_DIV_1
  – LL_RCC_PLL1MBOOST_DIV_2
  – LL_RCC_PLL1MBOOST_DIV_4
  – LL_RCC_PLL1MBOOST_DIV_6
  – LL_RCC_PLL1MBOOST_DIV_8
  – LL_RCC_PLL1MBOOST_DIV_10
  – LL_RCC_PLL1MBOOST_DIV_12
  – LL_RCC_PLL1MBOOST_DIV_14
  – LL_RCC_PLL1MBOOST_DIV_16

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1MBOOST LL_RCC_SetPll1EPodPrescaler

LL_RCC_GetPll1EPodPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetPll1EPodPrescaler (void)

Function description

Get PLL1 EPOD Prescaler booster input clock.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_PLL1MBOOST_DIV_1
  – LL_RCC_PLL1MBOOST_DIV_2
  – LL_RCC_PLL1MBOOST_DIV_4
  – LL_RCC_PLL1MBOOST_DIV_6
  – LL_RCC_PLL1MBOOST_DIV_8
  – LL_RCC_PLL1MBOOST_DIV_10
  – LL_RCC_PLL1MBOOST_DIV_12
  – LL_RCC_PLL1MBOOST_DIV_14
  – LL_RCC_PLL1MBOOST_DIV_16

Reference Manual to LL API cross reference:

- PLL1CFGR PLL1MBOOST LL_RCC_GetPll1EPodPrescaler

LL_RCC_PLL2_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Enable (void)

Function description

Enable PLL2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL2ON LL_RCC_PLL2_Enable

LL_RCC_PLL2_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2_Disable (void)

Function description

Disable PLL2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL2ON LL_RCC_PLL2_Disable

LL_RCC_PLL2_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsReady (void)

Function description

Check if PLL2 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL2RDY LL_RCC_PLL2_IsReady

LL_RCC_PLL2_ConfigDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL2_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)

Function description

Configure PLL2 used for 48Mhz domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLL2SOURCE_NONE
  - LL_RCC_PLL2SOURCE_MSIS
  - LL_RCC_PLL2SOURCE_HSI
  - LL_RCC_PLL2SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLQ:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL2 Source, PLLM, PLLN and PLLQ can be written only when PLL2 is disabled.
- This can be selected for USB, SDMMC

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2SRC LL_RCC_PLL2_ConfigDomain_48M
- PLL2CFGR PLL2M LL_RCC_PLL2_ConfigDomain_48M
- PLL2CFGR PLL2N LL_RCC_PLL2_ConfigDomain_48M
- PLL2CFGR PLL2Q LL_RCC_PLL2_ConfigDomain_48M

LL_RCC_PLL2_ConfigDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL2_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)

Function description

Configure PLL2 used for SAI domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_PLL2SOURCE_NONE
  – LL_RCC_PLL2SOURCE_MSIS
  – LL_RCC_PLL2SOURCE_HSI
  – LL_RCC_PLL2SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLP:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL1 Source, PLLM, PLLN and PLL2P can be written only when PLL1 is disabled.
- This can be selected for SAI1 or SAI2

Reference Manual to LL API cross reference:

- PLLC2FGR PLL2SRC LL_RCC_PLL2_ConfigDomain_SAI
- PLLC2FGR PLL2M LL_RCC_PLL2_ConfigDomain_SAI
- PLLC2FGR PLL2N LL_RCC_PLL2_ConfigDomain_SAI
- PLLC2FGR PLL2P LL_RCC_PLL2_ConfigDomain_SAI

LL_RCC_PLL2_ConfigDomain_ADC

Function name

__STATIC_INLINE void LL_RCC_PLL2_ConfigDomain_ADC (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)

Function description

Configure PLL2 used for ADC domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_PLL2SOURCE_NONE
  – LL_RCC_PLL2SOURCE_MSIS
  – LL_RCC_PLL2SOURCE_HSI
  – LL_RCC_PLL2SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLR:** parameter can be a value between 1 and 128
- **PLLN:** parameter can be a value between 4 and 512

Return values

- **None:**

Notes

- PLL2SRC/PLL2M/PLL2N/PLL2R can be written only when PLL2 is disabled.
- This can be selected for ADC

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2SRC LL_RCC_PLL2_ConfigDomain_ADC
- PLL2CFGR PLL2M LL_RCC_PLL2_ConfigDomain_ADC
- PLL2CFGR PLL2N LL_RCC_PLL2_ConfigDomain_ADC
- PLL2CFGR PLL2R LL_RCC_PLL2_ConfigDomain_ADC

LL_RCC_PLL2_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetSource (uint32_t PLL2Source)

Function description

Configure PLL2 clock source.

Parameters

- **PLL2Source:** This parameter can be one of the following values:
  - LL_RCC_PLL2SOURCE_NONE
  - LL_RCC_PLL2SOURCE_MSIS
  - LL_RCC_PLL2SOURCE_HSI
  - LL_RCC_PLL2SOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2SRC LL_RCC_PLL2_SetSource

LL_RCC_PLL2_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetSource (void)

Function description

Get the oscillator used as PLL2 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLL2SOURCE_NONE
  - LL_RCC_PLL2SOURCE_MSIS
  - LL_RCC_PLL2SOURCE_HSI
  - LL_RCC_PLL2SOURCE_HSE

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2SRC LL_RCC_PLL2_GetSource

LL_RCC_PLL2_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetN (uint32_t PLL2N)

Function description

Set Main PLL2 multiplication factor for VCO.

Parameters

- **PLL2N:** parameter can be a value between 4 and 512

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2N LL_RCC_PLL2_SetN

LL_RCC_PLL2_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetN (void)

Function description

Get Main PLL2 multiplication factor for VCO.

Return values

- **Between:** 4 and 512

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2N LL_RCC_PLL2_GetN

LL_RCC_PLL2_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetP (uint32_t PLL2P)

Function description

Set Main PLL2 division factor for PLL2P.

Parameters

- **PLL2P:** parameter can be a value between 1 and 128

Notes

- Used for SAI1 and SAI2 clock

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2P LL_RCC_PLL2_SetP

LL_RCC_PLL2_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetP (void)

Function description

Get Main PLL2 division factor for PLL2P.

Return values

- **Between:** 1 and 128

Notes

- Used for SAI1 and SAI2 clock

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2P LL_RCC_PLL2_GetP

LL_RCC_PLL2_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetQ (uint32_t PLL2Q)

Function description

Set Main PLL division factor for PLLQ.

Parameters

- **PLL2Q:** parameter can be a value between 1 and 128

Notes

- Used for PLL48M1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLLCFGR PLL2Q LL_RCC_PLL2_SetQ

LL_RCC_PLL2_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetQ (void)

Function description

Get Main PLL division factor for PLLQ.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL48M1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2Q LL_RCC_PLL2_GetQ

LL_RCC_PLL2_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetR (uint32_t PLL2R)

Function description

Set Main PLL division factor for PLLQ.

Parameters

- **PLL2R:** parameter can be a value between 1 and 128

Notes

- Used for PLL2CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2R LL_RCC_PLL2_SetR

LL_RCC_PLL2_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetR (void)

Function description

Get Main PLL2 division factor for PLL2R.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL2CLK (system clock)

Reference Manual to LL API cross reference:

- PLL2DIVR PLL2R LL_RCC_PLL2_GetR

LL_RCC_PLL2_SetDivider

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetDivider (uint32_t PLL2M)

Function description

Set Division factor for the main PLL and other PLL.

Parameters

- **PLL2M:** parameter can be a value between 1 and 16

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2M LL_RCC_PLL2_SetDivider

LL_RCC_PLL2_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetDivider (void)

Function description

Get Division factor for the main PLL and other PLL.

Return values

- **Between:** 1 and 16

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2M LL_RCC_PLL2_GetDivider

LL_RCC_PLL2_EnableDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableDomain_SAI (void)

Function description

Enable PLL2 output mapped on SAI domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2PEN LL_RCC_PLL2_EnableDomain_SAI

LL_RCC_PLL2_DisableDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableDomain_SAI (void)

Function description

Disable PLL2 output mapped on SAI domain clock.

Return values

- **None:**

Notes

- In order to save power, when of the PLL2 is not used, should be 0

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2PEN LL_RCC_PLL2_DisableDomain_SAI

LL_RCC_PLL2_IsEnabledDomain_SAI

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledDomain_SAI (void)

Function description

Check if PLL2 output mapped on SAI domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2PEN LL_RCC_PLL2_IsEnabledDomain_SAI

LL_RCC_PLL2_EnableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableDomain_48M (void)

Function description

Enable PLL2 output mapped on 48MHz domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2QEN LL_RCC_PLL2_EnableDomain_48M

LL_RCC_PLL2_DisableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableDomain_48M (void)

Function description

Disable PLL2 output mapped on 48MHz domain clock.

Return values

- **None:**

Notes

- In order to save power, when of the PLL2 48M is not used, should be 0

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2QEN LL_RCC_PLL2_DisableDomain_48M

LL_RCC_PLL2_IsEnabledDomain_48M

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledDomain_48M (void)

Function description

Check if PLL2 output mapped on 48M domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2QEN LL_RCC_PLL2_IsEnabledDomain_48M

LL_RCC_PLL2_EnableDomain_ADC

Function name

__STATIC_INLINE void LL_RCC_PLL2_EnableDomain_ADC (void)

Function description

Enable PLL2 output mapped on ADC domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2REN LL_RCC_PLL2_EnableDomain_ADC

LL_RCC_PLL2_DisableDomain_ADC

Function name

__STATIC_INLINE void LL_RCC_PLL2_DisableDomain_ADC (void)

Function description

Disable PLL2 output mapped on ADC domain clock.

Return values

- **None:**

Notes

- In order to save power, when of the PLL2 ADC is not used, Main PLL2ADC should be 0

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2REN LL_RCC_PLL2_DisableDomain_ADC

LL_RCC_PLL2_IsEnabledDomain_ADC

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_IsEnabledDomain_ADC (void)

Function description

Check if PLL2 output mapped on ADC domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2REN LL_RCC_PLL2_IsEnabledDomain_ADC

LL_RCC_PLL2FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL2FRACN_Enable (void)

Function description

Enable PLL2 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2FRACEN LL_RCC_PLL2FRACN_Enable

LL_RCC_PLL2FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2FRACN_IsEnabled (void)

Function description

Check if PLL2 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2FRACEN LL_RCC_PLL2FRACN_IsEnabled

LL_RCC_PLL2FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL2FRACN_Disable (void)

Function description

Disable PLL2 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2FRACEN LL_RCC_PLL2FRACN_Disable

LL_RCC_PLL2_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetFRACN (uint32_t FRACN)

Function description

Set PLL2 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL2FRACR PLL2FRACN LL_RCC_PLL2_SetFRACN

LL_RCC_PLL2_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL2_GetFRACN (void)

Function description

Get PLL2 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL2FRACR PLL2FRACN LL_RCC_PLL2_GetFRACN

LL_RCC_PLL2_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL2_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL2 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL2 is disabled.

Reference Manual to LL API cross reference:

- PLL2CFGR PLL2RGE LL_RCC_PLL2_SetVCOInputRange

LL_RCC_PLL3_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Enable (void)

Function description

Enable PLL3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL3ON LL_RCC_PLL3_Enable

LL_RCC_PLL3_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3_Disable (void)

Function description

Disable PLL3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PLL3ON LL_RCC_PLL3_Disable

LL_RCC_PLL3_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsReady (void)

Function description

Check if PLL3 Ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR PLL3RDY LL_RCC_PLL3_IsReady

LL_RCC_PLL3_ConfigDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL3_ConfigDomain_SAI (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLP)

Function description

Configure PLL3 used for SAI domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_MSIS
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLP:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL3SRC/PLL3M/PLL3N/PLL3PDIV can be written only when PLL3 is disabled.
- This can be selected for SAI1 or SAI2

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_ConfigDomain_SAI
- PLL3CFGR PLL3M LL_RCC_PLL3_ConfigDomain_SAI
- PLL3CFGR PLL3N LL_RCC_PLL3_ConfigDomain_SAI
- PLL3DIVR PLL3P LL_RCC_PLL3_ConfigDomain_SAI

LL_RCC_PLL3_ConfigDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL3_ConfigDomain_48M (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLQ)

Function description

Configure PLL3 used for 48Mhz domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_MSIS
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLQ:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL3 Source, PLLM, PLLN and PLLQ can be written only when PLL3 is disabled.
- This can be selected for USB, SDMMC

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_ConfigDomain_48M
- PLL3CFGR PLL3M LL_RCC_PLL3_ConfigDomain_48M
- PLL3CFGR PLL3N LL_RCC_PLL3_ConfigDomain_48M
- PLL3CFGR PLL3Q LL_RCC_PLL3_ConfigDomain_48M

LL_RCC_PLL3_ConfigDomain_HSPI_LTDC

Function name

__STATIC_INLINE void LL_RCC_PLL3_ConfigDomain_HSPI_LTDC (uint32_t Source, uint32_t PLLM, uint32_t PLLN, uint32_t PLLR)

Function description

Configure PLL3 used for HSPI_LTDC domain clock.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_MSIS
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE
- **PLLM:** parameter can be a value between 1 and 16
- **PLLN:** parameter can be a value between 4 and 512
- **PLLR:** parameter can be a value between 1 and 128

Return values

- **None:**

Notes

- PLL3 Source, PLLM, PLLN and PLLR can be written only when PLL3 is disabled.
- This can be selected for HSPI and LTDC

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
- PLL3CFGR PLL3M LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
- PLL3CFGR PLL3N LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
- PLL3CFGR PLL3R LL_RCC_PLL3_ConfigDomain_HSPI_LTDC

LL_RCC_PLL3_SetSource

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetSource (uint32_t PLLSource)

Function description

Configure PLL3 clock source.

Parameters

- **PLLSource:** This parameter can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_MSIS
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_SetSource

LL_RCC_PLL3_GetSource

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetSource (void)

Function description

Get the oscillator used as PLL3 clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_PLL3SOURCE_NONE
  - LL_RCC_PLL3SOURCE_MSIS
  - LL_RCC_PLL3SOURCE_HSI
  - LL_RCC_PLL3SOURCE_HSE

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3SRC LL_RCC_PLL3_GetSource

LL_RCC_PLL3_SetN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetN (uint32_t PLL3N)

Function description

Set Main PLL3 multiplication factor for VCO.

Parameters

- **PLL3N:** parameter can be a value between 4 and 512

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3N LL_RCC_PLL3_SetN

LL_RCC_PLL3_GetN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetN (void)

Function description

Get Main PLL3 multiplication factor for VCO.

Return values

- **Between:** 4 and 512

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3N LL_RCC_PLL3_GetN

LL_RCC_PLL3_SetP

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetP (uint32_t PLL3P)

Function description

Set Main PLL3 division factor for PLL3P.

Parameters

- **PLL3P:** parameter can be a value between 1 and 128

Notes

- Used for SAI1 and SAI2 clock

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3P LL_RCC_PLL3_SetP

LL_RCC_PLL3_GetP

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetP (void)

Function description

Get Main PLL3 division factor for PLL3P.

Return values

- **Between:** 1 and 128

Notes

- Used for SAI1 and SAI2 clock

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3P LL_RCC_PLL3_GetP

LL_RCC_PLL3_SetQ

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetQ (uint32_t PLL3Q)

Function description

Set Main PLL division factor for PLLQ.

Parameters

- **PLL3Q:** parameter can be a value between 1 and 128

Notes

- Used for PLL48M1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLLCFGR PLL3Q LL_RCC_PLL3_SetQ

LL_RCC_PLL3_GetQ

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetQ (void)

Function description

Get Main PLL division factor for PLLQ.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL48M1CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3Q LL_RCC_PLL3_GetQ

LL_RCC_PLL3_SetR

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetR (uint32_t PLL3R)

Function description

Set Main PLL division factor for PLLQ.

Parameters

- **PLL3R:** parameter can be a value between 1 and 128

Notes

- Used for PLL3CLK selected for USB, SDMMC (48 MHz clock)

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3R LL_RCC_PLL3_SetR

LL_RCC_PLL3_GetR

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetR (void)

Function description

Get Main PLL3 division factor for PLL3R.

Return values

- **Between:** 1 and 128

Notes

- Used for PLL3CLK (system clock)

Reference Manual to LL API cross reference:

- PLL3DIVR PLL3R LL_RCC_PLL3_GetR

LL_RCC_PLL3_SetDivider

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetDivider (uint32_t PLL3M)

Function description

Set Division factor for the main PLL and other PLL.

Parameters

- **PLL3M:** parameter can be a value between 1 and 16

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3M LL_RCC_PLL3_SetDivider

LL_RCC_PLL3_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetDivider (void)

Function description

Get Division factor for the main PLL and other PLL.

Return values

- **Between:** 1 and 16

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3M LL_RCC_PLL3_GetDivider

LL_RCC_PLL3_EnableDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableDomain_SAI (void)

Function description

Enable PLL3 output mapped on SAI domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3PEN LL_RCC_PLL3_EnableDomain_SAI

LL_RCC_PLL3_DisableDomain_SAI

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableDomain_SAI (void)

Function description

Disable PLL3 output mapped on SAI domain clock.

Return values

- **None:**

Notes

- In order to save power, when of the PLLSAI2 is not used, should be 0

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3PEN LL_RCC_PLL3_DisableDomain_SAI

LL_RCC_PLL3_IsEnabledDomain_SAI

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledDomain_SAI (void)

Function description

Check if PLL3 output mapped on SAI domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3PEN LL_RCC_PLL3_IsEnabledDomain_SAI

LL_RCC_PLL3_EnableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableDomain_48M (void)

Function description

Enable PLL2 output mapped on 48MHz domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3QEN LL_RCC_PLL3_EnableDomain_48M

LL_RCC_PLL3_DisableDomain_48M

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableDomain_48M (void)

Function description

Disable PLL3 output mapped on 48MHz domain clock.

Return values

- **None:**

Notes

- In order to save power, when of the PLL3 is not used, should be 0

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3QEN LL_RCC_PLL3_DisableDomain_48M

LL_RCC_PLL3_IsEnabledDomain_48M

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledDomain_48M (void)

Function description

Check if PLL3 output mapped on 48M domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3QEN LL_RCC_PLL3_IsEnabledDomain_48M

LL_RCC_PLL3_EnableDomain_HSPI_LTDC

Function name

__STATIC_INLINE void LL_RCC_PLL3_EnableDomain_HSPI_LTDC (void)

Function description

Enable PLL3 output mapped on HSPI_LTDC domain clock.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3REN LL_RCC_PLL3_EnableDomain_HSPI_LTDC

LL_RCC_PLL3_DisableDomain_HSPI_LTDC

Function name

__STATIC_INLINE void LL_RCC_PLL3_DisableDomain_HSPI_LTDC (void)

Function description

Disable PLL3 output mapped on HSPI_LTDC domain clock.

Return values

- **None:**

Notes

- In order to save power, when of the PLL3 is not used, should be 0

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3REN LL_RCC_PLL3_DisableDomain_HSPI_LTDC

LL_RCC_PLL3_IsEnabledDomain_HSPI_LTDC

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_IsEnabledDomain_HSPI_LTDC (void)

Function description

Check if PLL3 output mapped on HSPI_LTDC domain clock is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3REN LL_RCC_PLL3_IsEnabledDomain_HSPI_LTDC

LL_RCC_PLL3FRACN_Enable

Function name

__STATIC_INLINE void LL_RCC_PLL3FRACN_Enable (void)

Function description

Enable PLL3 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3FRACEN LL_RCC_PLL3FRACN_Enable

LL_RCC_PLL3FRACN_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3FRACN_IsEnabled (void)

Function description

Check if PLL3 FRACN is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3FRACEN LL_RCC_PLL3FRACN_IsEnabled

LL_RCC_PLL3FRACN_Disable

Function name

__STATIC_INLINE void LL_RCC_PLL3FRACN_Disable (void)

Function description

Disable PLL3 FRACN.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3FRACEN LL_RCC_PLL3FRACN_Disable

LL_RCC_PLL3_SetFRACN

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetFRACN (uint32_t FRACN)

Function description

Set PLL3 FRACN Coefficient.

Parameters

- **FRACN:** parameter can be a value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL3FRACR PLL3FRACN LL_RCC_PLL3_SetFRACN

LL_RCC_PLL3_GetFRACN

Function name

__STATIC_INLINE uint32_t LL_RCC_PLL3_GetFRACN (void)

Function description

Get PLL3 FRACN Coefficient.

Return values

- **A:** value between 0 and 8191 (0x1FFF)

Reference Manual to LL API cross reference:

- PLL3FRACR PLL3FRACN LL_RCC_PLL3_GetFRACN

LL_RCC_PLL3_SetVCOInputRange

Function name

__STATIC_INLINE void LL_RCC_PLL3_SetVCOInputRange (uint32_t InputRange)

Function description

Set PLL3 VCO Input Range.

Parameters

- **InputRange:** This parameter can be one of the following values:
  - LL_RCC_PLLINPUTRANGE_4_8
  - LL_RCC_PLLINPUTRANGE_8_16

Return values

- **None:**

Notes

- This API shall be called only when PLL3 is disabled.

Reference Manual to LL API cross reference:

- PLL3CFGR PLL3RGE LL_RCC_PLL3_SetVCOInputRange

LL_RCC_EnableSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_EnableSecPrivilegedMode (void)

Function description

Enable Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_EnableSecPrivilegedMode

LL_RCC_DisableSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_DisableSecPrivilegedMode (void)

Function description

Disable Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_DisableSecPrivilegedMode

LL_RCC_IsEnabledSecPrivilegedMode

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledSecPrivilegedMode (void)

Function description

Check if Secure Privileged mode has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_IsEnabledSecPrivilegedMode

LL_RCC_EnableNSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_EnableNSecPrivilegedMode (void)

Function description

Enable Non Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_RCC_EnableNSecPrivilegedMode

LL_RCC_DisableNSecPrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_DisableNSecPrivilegedMode (void)

Function description

Disable Non Secure Privileged mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_RCC_DisableNSecPrivilegedMode

LL_RCC_IsEnabledNSecPrivilegedMode

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledNSecPrivilegedMode (void)

Function description

Check if Non Secure Privileged mode has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- PRIVCFGR NSPRIV LL_RCC_IsEnabledNSecPrivilegedMode

LL_RCC_EnablePrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_EnablePrivilegedMode (void)

Function description

Enable privileged mode.

Return values

- **None:**

Notes

- User should use LL_RCC_EnableSecPrivilegedMode() to enable Secure privilege User should use LL_RCC_EnableNSecPrivilegedMode() to enable Non-secure privilege This API is kept for legacy purpose only

Reference Manual to LL API cross reference:

- PRIVCFGR SPRIV LL_RCC_EnablePrivilegedMode
- PRIVCFGR NSPRIV LL_RCC_EnablePrivilegedMode

LL_RCC_DisablePrivilegedMode

Function name

__STATIC_INLINE void LL_RCC_DisablePrivilegedMode (void)

Function description

Disable Privileged mode.

Return values

- **None:**

Notes

- User should use LL_RCC_DisableSecPrivilegedMode() to disable Secure privilege User should use LL_RCC_DisableNSecPrivilegedMode() to disable Non-secure privilege This API is kept for legacy purpose only

Reference Manual to LL API cross reference:

- CR PRIV LL_RCC_DisablePrivilegedMode

LL_RCC_IsEnabledPrivilegedMode

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledPrivilegedMode (void)

Function description

Check if Privileged mode has been enabled or not.

Return values

- **State:** of bit (1 or 0).

Notes

- User should use LL_RCC_IsEnabledSecPrivilegedMode() to check Secure privilege setting User should use LL_RCC_IsEnabledNSecPrivilegedMode() to check Non-secure privilege setting This API is kept for legacy purpose only

Reference Manual to LL API cross reference:

- CR PRIV LL_RCC_IsEnabledPrivilegedMode

LL_RCC_ClearFlag_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSIRDY (void)

Function description

Clear LSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSIRDYC LL_RCC_ClearFlag_LSIRDY

LL_RCC_ClearFlag_LSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSERDY (void)

Function description

Clear LSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSERDYC LL_RCC_ClearFlag_LSERDY

LL_RCC_ClearFlag_MSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_MSIRDY (void)

Function description

Clear MSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR MSISRDYC LL_RCC_ClearFlag_MSIRDY

LL_RCC_ClearFlag_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSIRDY (void)

Function description

Clear HSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSIRDYC LL_RCC_ClearFlag_HSIRDY

LL_RCC_ClearFlag_HSERDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSERDY (void)

Function description

Clear HSE ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSERDYC LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSI48RDY (void)

Function description

Clear HSI48 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR HSI48RDYC LL_RCC_ClearFlag_HSI48RDY

LL_RCC_ClearFlag_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL1RDY (void)

Function description

Clear PLL1 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL1RDYC LL_RCC_ClearFlag_PLL1RDY

LL_RCC_ClearFlag_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL2RDY (void)

Function description

Clear PLL2 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL2RDYC LL_RCC_ClearFlag_PLL2RDY

LL_RCC_ClearFlag_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_PLL3RDY (void)

Function description

Clear PLL3 ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR PLL3RDYC LL_RCC_ClearFlag_PLL3RDY

LL_RCC_ClearFlag_HSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_HSECSS (void)

Function description

Clear Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR CSSC LL_RCC_ClearFlag_HSECSS

LL_RCC_ClearFlag_MSIKRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_MSIKRDY (void)

Function description

Clear MSIK ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR MSIKRDYC LL_RCC_ClearFlag_MSIKRDY

LL_RCC_ClearFlag_SHSIRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_SHSIRDY (void)

Function description

Clear SHSI ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR SHSIRDYC LL_RCC_ClearFlag_SHSIRDY

LL_RCC_IsActiveFlag_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSIRDY (void)

Function description

Check if LSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSIRDYF LL_RCC_IsActiveFlag_LSIRDY

LL_RCC_IsActiveFlag_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSERDY (void)

Function description

Check if LSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSERDYF LL_RCC_IsActiveFlag_LSERDY

LL_RCC_IsActiveFlag_MSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_MSIRDY (void)

Function description

Check if MSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR MSISRDYF LL_RCC_IsActiveFlag_MSIRDY

LL_RCC_IsActiveFlag_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSIRDY (void)

Function description

Check if HSI ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSIRDYF LL_RCC_IsActiveFlag_HSIRDY

LL_RCC_IsActiveFlag_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSERDY (void)

Function description

Check if HSE ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSERDYF LL_RCC_IsActiveFlag_HSERDY

LL_RCC_IsActiveFlag_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSI48RDY (void)

Function description

Check if HSI48 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR HSI48RDYF LL_RCC_IsActiveFlag_HSI48RDY

LL_RCC_IsActiveFlag_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL1RDY (void)

Function description

Check if PLL1 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL1RDYF LL_RCC_IsActiveFlag_PLL1RDY

LL_RCC_IsActiveFlag_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL2RDY (void)

Function description

Check if PLL2 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL2RDYF LL_RCC_IsActiveFlag_PLL2RDY

LL_RCC_IsActiveFlag_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PLL3RDY (void)

Function description

Check if PLL3 ready interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR PLL3RDYF LL_RCC_IsActiveFlag_PLL3RDY

LL_RCC_IsActiveFlag_HSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_HSECSS (void)

Function description

Check if Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR CSSF LL_RCC_IsActiveFlag_HSECSS

LL_RCC_IsActiveFlag_MSIKRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_MSIKRDY (void)

Function description

Check if Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR MSIKRDYF LL_RCC_IsActiveFlag_MSIKRDY

LL_RCC_IsActiveFlag_SHSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SHSIRDY (void)

Function description

Check if Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR SHSIRDYF LL_RCC_IsActiveFlag_SHSIRDY

LL_RCC_IsActiveFlag_IWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_IWDGRST (void)

Function description

Check if RCC flag Independent Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR IWDGRSTF LL_RCC_IsActiveFlag_IWDGRST

LL_RCC_IsActiveFlag_LPWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPWRRST (void)

Function description

Check if RCC flag Low Power reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR LPWRRSTF LL_RCC_IsActiveFlag_LPWRRST

LL_RCC_IsActiveFlag_OBLRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_OBLRST (void)

Function description

Check if RCC flag is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR OBLRSTF LL_RCC_IsActiveFlag_OBLRST

LL_RCC_IsActiveFlag_PINRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PINRST (void)

Function description

Check if RCC flag Pin reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR PINRSTF LL_RCC_IsActiveFlag_PINRST

LL_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_WWDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WWDGRST (void)

Function description

Check if RCC flag Window Watchdog reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_BORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_BORRST (void)

Function description

Check if RCC flag BOR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR BORRSTF LL_RCC_IsActiveFlag_BORRST

LL_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RMVF LL_RCC_ClearResetFlags

LL_RCC_EnableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSIRDY (void)

Function description

Enable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_EnableIT_LSIRDY

LL_RCC_EnableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LSERDY (void)

Function description

Enable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_EnableIT_LSERDY

LL_RCC_EnableIT_MSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_MSIRDY (void)

Function description

Enable MSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER MSIRDYIE LL_RCC_EnableIT_MSIRDY

LL_RCC_EnableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSIRDY (void)

Function description

Enable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_EnableIT_HSIRDY

LL_RCC_EnableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSERDY (void)

Function description

Enable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_EnableIT_HSERDY

LL_RCC_EnableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_HSI48RDY (void)

Function description

Enable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_EnableIT_HSI48RDY

LL_RCC_EnableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL1RDY (void)

Function description

Enable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_EnableIT_PLL1RDY

LL_RCC_EnableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL2RDY (void)

Function description

Enable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_EnableIT_PLL2RDY

LL_RCC_EnableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_PLL3RDY (void)

Function description

Enable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_EnableIT_PLL3RDY

LL_RCC_EnableIT_MSIKRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_MSIKRDY (void)

Function description

Enable MSIKRDYIE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER MSIKRDYIE LL_RCC_EnableIT_MSIKRDY

LL_RCC_EnableIT_SHSIRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_SHSIRDY (void)

Function description

Enable SHSIRDYIE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER SHSIRDYIE LL_RCC_EnableIT_SHSIRDY

LL_RCC_DisableIT_LSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSIRDY (void)

Function description

Disable LSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_DisableIT_LSIRDY

LL_RCC_DisableIT_LSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LSERDY (void)

Function description

Disable LSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_DisableIT_LSERDY

LL_RCC_DisableIT_MSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_MSIRDY (void)

Function description

Disable MSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER MSISRDYIE LL_RCC_DisableIT_MSIRDY

LL_RCC_DisableIT_HSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSIRDY (void)

Function description

Disable HSI ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_DisableIT_HSIRDY

LL_RCC_DisableIT_HSERDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSERDY (void)

Function description

Disable HSE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_DisableIT_HSERDY

LL_RCC_DisableIT_HSI48RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_HSI48RDY (void)

Function description

Disable HSI48 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_DisableIT_HSI48RDY

LL_RCC_DisableIT_PLL1RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL1RDY (void)

Function description

Disable PLL1 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_DisableIT_PLL1RDY

LL_RCC_DisableIT_PLL2RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL2RDY (void)

Function description

Disable PLL2 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_DisableIT_PLL2RDY

LL_RCC_DisableIT_PLL3RDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_PLL3RDY (void)

Function description

Disable PLL3 ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_DisableIT_PLL3RDY

LL_RCC_DisableIT_MSIKRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_MSIKRDY (void)

Function description

Disable MSIKRDYIE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER MSIKRDYIE LL_RCC_DisableIT_MSIKRDY

LL_RCC_DisableIT_SHSIRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_SHSIRDY (void)

Function description

Disable SHSIRDYIE ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER SHSIRDYIE LL_RCC_DisableIT_SHSIRDY

LL_RCC_IsEnabledIT_LSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSIRDY (void)

Function description

Checks if LSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSIRDYIE LL_RCC_IsEnabledIT_LSIRDY

LL_RCC_IsEnabledIT_LSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LSERDY (void)

Function description

Checks if LSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER LSERDYIE LL_RCC_IsEnabledIT_LSERDY

LL_RCC_IsEnabledIT_MSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_MSIRDY (void)

Function description

Checks if MSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER MSIRDYIE LL_RCC_IsEnabledIT_MSIRDY

LL_RCC_IsEnabledIT_HSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSIRDY (void)

Function description

Checks if HSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSIRDYIE LL_RCC_IsEnabledIT_HSIRDY

LL_RCC_IsEnabledIT_HSERDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSERDY (void)

Function description

Checks if HSE ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSERDYIE LL_RCC_IsEnabledIT_HSERDY

LL_RCC_IsEnabledIT_HSI48RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_HSI48RDY (void)

Function description

Checks if HSI48 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER HSI48RDYIE LL_RCC_IsEnabledIT_HSI48RDY

LL_RCC_IsEnabledIT_PLL1RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL1RDY (void)

Function description

Checks if PLL1 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL1RDYIE LL_RCC_IsEnabledIT_PLL1RDY

LL_RCC_IsEnabledIT_PLL2RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL2RDY (void)

Function description

Checks if PLL2 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL2RDYIE LL_RCC_IsEnabledIT_PLL2RDY

LL_RCC_IsEnabledIT_PLL3RDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_PLL3RDY (void)

Function description

Checks if PLL3 ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER PLL3RDYIE LL_RCC_IsEnabledIT_PLL3RDY

LL_RCC_IsEnabledIT_MSIKRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_MSIKRDY (void)

Function description

Checks if MSIK ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER MSIKRDYIE LL_RCC_IsEnabledIT_MSIKRDY

LL_RCC_IsEnabledIT_SHSIRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_SHSIRDY (void)

Function description

Checks if SHSI ready interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIER SHSIRDYIE LL_RCC_IsEnabledIT_SHSIRDY

LL_RCC_ConfigSecure

Function name

__STATIC_INLINE void LL_RCC_ConfigSecure (uint32_t SecureConfig)

Function description

Configure RCC resources security.

Parameters

- **SecureConfig:** This parameter can be one or a combination of the following values:
  - LL_RCC_ALL_NSEC & LL_RCC_ALL_SEC
  - LL_RCC_HSI_SEC & LL_RCC_HSI_NSEC
  - LL_RCC_HSE_SEC & LL_RCC_HSE_NSEC
  - LL_RCC_MSI_SEC & LL_RCC_MSI_NSEC
  - LL_RCC_LSE_SEC & LL_RCC_LSE_NSEC
  - LL_RCC_LSI_SEC & LL_RCC_LSI_NSEC
  - LL_RCC_SYSCLK_SEC & LL_RCC_SYSCLK_NSEC
  - LL_RCC_PRESCALERS_SEC & LL_RCC_PRESCALERS_NSEC
  - LL_RCC_PLL1_SEC & LL_RCC_PLL1_NSEC
  - LL_RCC_PLL2_SEC & LL_RCC_PLL2_NSEC
  - LL_RCC_PLL3_SEC & LL_RCC_PLL3_NSEC
  - LL_RCC_CLK48M_SEC & LL_RCC_CLK48M_NSEC
  - LL_RCC_HSI48_SEC & LL_RCC_HSI48_NSEC
  - LL_RCC_RESET_FLAGS_SEC & LL_RCC_RESET_FLAGS_NSEC

Return values

- **None:**

Notes

- Only available from secure state when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR HSISEC LL_RCC_ConfigSecure
- SECCFGR HSESEC LL_RCC_ConfigSecure
- SECCFGR MSISEC LL_RCC_ConfigSecure
- SECCFGR LSISEC LL_RCC_ConfigSecure
- SECCFGR LSESEC LL_RCC_ConfigSecure
- SECCFGR SYSCLKSEC LL_RCC_ConfigSecure
- SECCFGR PRESCSEC LL_RCC_ConfigSecure
- SECCFGR PLL1SEC LL_RCC_ConfigSecure
- SECCFGR PLL2SEC LL_RCC_ConfigSecure
- SECCFGR PLL3SEC LL_RCC_ConfigSecure
- SECCFGR CLK48MSEC LL_RCC_ConfigSecure
- SECCFGR HSI48SEC LL_RCC_ConfigSecure
- SECCFGR RMVFSEC LL_RCC_ConfigSecure

LL_RCC_GetConfigSecure

Function name

__STATIC_INLINE uint32_t LL_RCC_GetConfigSecure (void)

Function description

Get RCC resources security status.

Return values

- **Returned:** value can be one or a combination of the following values:
  - LL_RCC_ALL_NSEC & LL_RCC_ALL_SEC
  - LL_RCC_HSI_SEC & LL_RCC_HSI_NSEC
  - LL_RCC_HSE_SEC & LL_RCC_HSE_NSEC
  - LL_RCC_MSI_SEC & LL_RCC_MSI_NSEC
  - LL_RCC_LSE_SEC & LL_RCC_LSE_NSEC
  - LL_RCC_LSI_SEC & LL_RCC_LSI_NSEC
  - LL_RCC_SYSCLK_SEC & LL_RCC_SYSCLK_NSEC
  - LL_RCC_PRESCALERS_SEC & LL_RCC_PRESCALERS_NSEC
  - LL_RCC_PLL1_SEC & LL_RCC_PLL1_NSEC
  - LL_RCC_PLL2_SEC & LL_RCC_PLL2_NSEC
  - LL_RCC_PLL3_SEC & LL_RCC_PLL3_NSEC
  - LL_RCC_CLK48M_SEC & LL_RCC_CLK48M_NSEC
  - LL_RCC_HSI48_SEC & LL_RCC_HSI48_NSEC
  - LL_RCC_RESET_FLAGS_SEC & LL_RCC_RESET_FLAGS_NSEC
- **None:**

Notes

- Only available from secure state when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR HSISEC LL_RCC_GetConfigSecure
- SECCFGR HSESEC LL_RCC_GetConfigSecure
- SECCFGR MSISEC LL_RCC_GetConfigSecure
- SECCFGR LSISEC LL_RCC_GetConfigSecure
- SECCFGR LSESEC LL_RCC_GetConfigSecure
- SECCFGR SYSCLKSEC LL_RCC_GetConfigSecure
- SECCFGR PRESCSEC LL_RCC_GetConfigSecure
- SECCFGR PLL1SEC LL_RCC_GetConfigSecure
- SECCFGR PLL2SEC LL_RCC_GetConfigSecure
- SECCFGR PLL3SEC LL_RCC_GetConfigSecure
- SECCFGR CLK48MSEC LL_RCC_GetConfigSecure
- SECCFGR HSI48SEC LL_RCC_GetConfigSecure
- SECCFGR RMVFSEC LL_RCC_GetConfigSecure

LL_RCC_DeInit

Function name

ErrorStatus LL_RCC_DeInit (void)

Function description

Reset the RCC clock configuration to the default reset state.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RCC registers are de-initialized
  - ERROR: not applicable

Notes

- The default reset state of the clock configuration is given below: MSI ON and used as system clock sourceHSE, HSI, PLL1, PLL2 and PLL3 OFFAHB, APB1, APB2 and APB3 prescaler set to 1.CSS, MCO OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)

Function description

Return the frequencies of different on chip clocks; System, AHB, APB1 and APB2 buses clocks.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

Notes

- Each time SYSCLK, HCLK, PCLK1, PCLK2 and/or PCLK3 clock changes, this function must be called to update structure fields. Otherwise, any configuration based on this function will be incorrect.

LL_RCC_GetUSARTClockFreq

Function name

uint32_t LL_RCC_GetUSARTClockFreq (uint32_t USARTxSource)

Function description

Return USARTx clock frequency.

Parameters

- **USARTxSource:** This parameter can be one of the following values:
  – LL_RCC_USART1_CLKSOURCE
  – LL_RCC_USART2_CLKSOURCE
  – LL_RCC_USART3_CLKSOURCE
  – LL_RCC_USART6_CLKSOURCE (*)

Return values

- **USART:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetUARTClockFreq

Function name

uint32_t LL_RCC_GetUARTClockFreq (uint32_t UARTxSource)

Function description

Return UARTx clock frequency.

Parameters

- **UARTxSource:** This parameter can be one of the following values:
  – LL_RCC_UART4_CLKSOURCE
  – LL_RCC_UART5_CLKSOURCE

Return values

- **UART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetI2CClockFreq

Function name

uint32_t LL_RCC_GetI2CClockFreq (uint32_t I2CxSource)

Function description

Return I2Cx clock frequency.

Parameters

- **I2CxSource:** This parameter can be one of the following values:
  - LL_RCC_I2C1_CLKSOURCE
  - LL_RCC_I2C2_CLKSOURCE
  - LL_RCC_I2C3_CLKSOURCE
  - LL_RCC_I2C4_CLKSOURCE

Return values

- **I2C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or MSIK) is not ready

LL_RCC_GetLPUARTClockFreq

Function name

uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)

Function description

Return LPUARTx clock frequency.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  - LL_RCC_LPUART1_CLKSOURCE

Return values

- **LPUART:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or LSE) is not ready

LL_RCC_GetLPTIMClockFreq

Function name

uint32_t LL_RCC_GetLPTIMClockFreq (uint32_t LPTIMxSource)

Function description

Return LPTIMx clock frequency.

Parameters

- **LPTIMxSource:** This parameter can be one of the following values:
  - LL_RCC_LPTIM1_CLKSOURCE
  - LL_RCC_LPTIM2_CLKSOURCE
  - LL_RCC_LPTIM34_CLKSOURCE

Return values

- **LPTIM:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI, LSI or LSE) is not ready

LL_RCC_GetFDCANClockFreq

Function name

uint32_t LL_RCC_GetFDCANClockFreq (uint32_t FDCANxSource)

Function description

Return FDCAN kernel clock frequency.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN_CLKSOURCE

Return values

- **FDCAN:** kernel clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator HSE or PLL is not ready

LL_RCC_GetSAIClockFreq

Function name

uint32_t LL_RCC_GetSAIClockFreq (uint32_t SAIxSource)

Function description

Return SAIx clock frequency.

Parameters

- **SAIxSource:** This parameter can be one of the following values:
  - LL_RCC_SAI1_CLKSOURCE
  - LL_RCC_SAI2_CLKSOURCE

Return values

- **SAI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that PLL is not ready

LL_RCC_GetSDMMCKernelClockFreq

Function name

uint32_t LL_RCC_GetSDMMCKernelClockFreq (uint32_t SDMMCxSource)

Function description

Return SDMMCx kernel clock frequency.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  - LL_RCC_SDMMC_KERNELCLKSOURCE

Return values

- **SDMMC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator or PLL is not ready

LL_RCC_GetSDMMCClockFreq

Function name

uint32_t LL_RCC_GetSDMMCClockFreq (uint32_t SDMMCxSource)

Function description

Return SDMMCx clock frequency.

Parameters

- **SDMMCxSource:** This parameter can be one of the following values:
  – LL_RCC_SDMMC_CLKSOURCE

Return values

- **SDMMC:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI /MSIK /HSI) or PLL is not ready

LL_RCC_GetRNGClockFreq

Function name

uint32_t LL_RCC_GetRNGClockFreq (uint32_t RNGxSource)

Function description

Return RNGx clock frequency.

Parameters

- **RNGxSource:** This parameter can be one of the following values:
  – LL_RCC_RNG_CLKSOURCE

Return values

- **RNG:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI or HSI48) or PLL is not ready

LL_RCC_GetUSBClockFreq

Function name

uint32_t LL_RCC_GetUSBClockFreq (uint32_t USBxSource)

Function description

Return USBx clock frequency.

Parameters

- **USBxSource:** This parameter can be one of the following values:
  – LL_RCC_USB_CLKSOURCE

Return values

- **USB:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI /HSI48) or PLL is not ready

LL_RCC_GetADCDACClockFreq

Function name

uint32_t LL_RCC_GetADCDACClockFreq (uint32_t ADCxSource)

Function description

Return ADCxDAC clock frequency.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADCDAC_CLKSOURCE

Return values

- **ADC/DAC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSI /HSI /MSIK) or PLL is not ready

LL_RCC_GetADF1ClockFreq

Function name

uint32_t LL_RCC_GetADF1ClockFreq (uint32_t ADF1Source)

Function description

Return ADF1 clock frequency.

Parameters

- **ADF1Source:** This parameter can be one of the following values:
  - LL_RCC_ADF1_CLKSOURCE

Return values

- **ADF1:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSIK) or PLL is not ready

LL_RCC_GetMDF1ClockFreq

Function name

uint32_t LL_RCC_GetMDF1ClockFreq (uint32_t MDF1Source)

Function description

Return MDF1 clock frequency.

Parameters

- **MDF1Source:** This parameter can be one of the following values:
  - LL_RCC_MDF1_CLKSOURCE

Return values

- **MDF1:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (MSIK) or PLL is not ready

LL_RCC_GetDAC1ClockFreq

Function name

uint32_t LL_RCC_GetDAC1ClockFreq (uint32_t DAC1Source)

Function description

Return DAC1 clock frequency.

Parameters

- **DAC1Source:** This parameter can be one of the following values:
  - LL_RCC_DAC1_CLKSOURCE

Return values

- **DAC1:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that LSI or LSE oscillator is not ready

LL_RCC_GetOCTOSPIClockFreq

Function name

uint32_t LL_RCC_GetOCTOSPIClockFreq (uint32_t OCTOSPIxSource)

Function description

Return OCTOSPI clock frequency.

Parameters

- **OCTOSPIxSource:** This parameter can be one of the following values:
  – LL_RCC_OCTOSPI_CLKSOURCE

Return values

- **OCTOSPI:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator MSIK or PLL is not ready

LL_RCC_GetSAESClockFreq

Function name

uint32_t LL_RCC_GetSAESClockFreq (uint32_t SAESxSource)

Function description

Return SAESx clock frequency.

Parameters

- **SAESxSource:** This parameter can be one of the following values:
  – LL_RCC_SAES_CLKSOURCE

Return values

- **SAEx:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator SHSI is not ready

LL_RCC_GetSPIClockFreq

Function name

uint32_t LL_RCC_GetSPIClockFreq (uint32_t SPIxSource)

Function description

Return SPIx clock frequency.

Parameters

- **SPIxSource:** This parameter can be one of the following values:
  – LL_RCC_SPI1_CLKSOURCE
  – LL_RCC_SPI2_CLKSOURCE
  – LL_RCC_SPI3_CLKSOURCE

Return values

- **SPI:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI or MSIK) is not ready

LL_RCC_GetDSIClockFreq

Function name

uint32_t LL_RCC_GetDSIClockFreq (uint32_t DSIxSource)

Function description

Return DSI clock frequency.

Parameters

- **DSIxSource:** This parameter can be one of the following values:
  - LL_RCC_DSI_CLKSOURCE

Return values

- **DSI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that external clock is used

LL_RCC_GetHSPIClockFreq

Function name

uint32_t LL_RCC_GetHSPIClockFreq (uint32_t HSPIxSource)

Function description

Return HSPI clock frequency.

Parameters

- **HSPIxSource:** This parameter can be one of the following values:
  - LL_RCC_HSPI_CLKSOURCE

Return values

- **HSPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetLTDCClockFreq

Function name

uint32_t LL_RCC_GetLTDCClockFreq (uint32_t LTDCxSource)

Function description

Return LTDC clock frequency.

Parameters

- **LTDCxSource:** This parameter can be one of the following values:
  - LL_RCC_LTDC_CLKSOURCE

Return values

- **HSPI:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

114.3 RCC Firmware driver defines

The following section lists the various define and macros of the module.

114.3.1 RCC

RCC

Peripheral ADCDAC get clock source

LL_RCC_ADCDAC_CLKSOURCE

ADCDACs Clock source selection

Peripheral ADCx and DAC1 clock source selection

LL_RCC_ADCDAC_CLKSOURCE_HCLK

No clock used as ADCx/DAC1 clock source

LL_RCC_ADCDAC_CLKSOURCE_SYSCLK

SYSCLK clock used as ADCx/DAC1 clock source

LL_RCC_ADCDAC_CLKSOURCE_PLL2

PLL2 clock used as ADCx/DAC1 clock source

LL_RCC_ADCDAC_CLKSOURCE_HSI

HSI clock used as ADCx/DAC1 clock source

LL_RCC_ADCDAC_CLKSOURCE_HSE

HSE clock used as ADCx/DAC1 clock source

LL_RCC_ADCDAC_CLKSOURCE_MSIK

MSIK clock used as ADCx/DAC1 clock source

Peripheral ADF1 get clock source

LL_RCC_ADF1_CLKSOURCE

ADF1 Clock source selection

Peripheral ADF1 clock source selection

LL_RCC_ADF1_CLKSOURCE_HCLK

HCLK clock used as ADF1 clock

LL_RCC_ADF1_CLKSOURCE_PLL1

PLL1 clock used as ADF1 clock

LL_RCC_ADF1_CLKSOURCE_PLL3

PLL3 clock used as ADF1 clock

LL_RCC_ADF1_CLKSOURCE_MSIK

MSIK clock used as ADF1 clock

LL_RCC_ADF1_CLKSOURCE_PIN

PIN SAI1_EXTCLK clock used as ADF1 clock

APB low-speed prescaler (APB1)

LL_RCC_APB1_DIV_1

HCLK not divided

LL_RCC_APB1_DIV_2

HCLK divided by 2

LL_RCC_APB1_DIV_4

HCLK divided by 4

LL_RCC_APB1_DIV_8

HCLK divided by 8

LL_RCC_APB1_DIV_16

HCLK divided by 16

APB high-speed prescaler (APB2)

LL_RCC_APB2_DIV_1
HCLK not divided

LL_RCC_APB2_DIV_2
HCLK divided by 2

LL_RCC_APB2_DIV_4
HCLK divided by 4

LL_RCC_APB2_DIV_8
HCLK divided by 8

LL_RCC_APB2_DIV_16
HCLK divided by 16

APB high-speed prescaler (APB3)

LL_RCC_APB3_DIV_1
HCLK not divided

LL_RCC_APB3_DIV_2
HCLK divided by 2

LL_RCC_APB3_DIV_4
HCLK divided by 4

LL_RCC_APB3_DIV_8
HCLK divided by 8

LL_RCC_APB3_DIV_16
HCLK divided by 16

Peripheral DAC1 get clock source

LL_RCC_DAC1_CLKSOURCE

Peripheral DAC1 clock source selection

LL_RCC_DAC1_CLKSOURCE_LSE
LSE clock used as DAC1 clock

LL_RCC_DAC1_CLKSOURCE_LSI
LSI clock used as DAC1 clock

DSI PHY clock prescaler (DCLK)

LL_RCC_DPHY_DIV_1
DCLK not divided

LL_RCC_DPHY_DIV_2
DCLK divided by 2

LL_RCC_DPHY_DIV_4
DCLK divided by 4

LL_RCC_DPHY_DIV_8
DCLK divided by 8

LL_RCC_DPHY_DIV_16

DCLK divided by 16

Peripheral DSI get clock source

LL_RCC_DSI_CLKSOURCE

Peripheral DSI clock source selection

LL_RCC_DSI_CLKSOURCE_PHY

LL_RCC_DSI_CLKSOURCE_PLL3

Peripheral FDCAN get kernel clock source

LL_RCC_FDCAN_CLKSOURCE

FDCAN Kernel Clock source selection

Peripheral FDCAN kernel clock source selection

LL_RCC_FDCAN_CLKSOURCE_HSE

HSE clock used as FDCAN kernel clock source

LL_RCC_FDCAN_CLKSOURCE_PLL1

PLL1 Q clock used as FDCAN kernel clock source

LL_RCC_FDCAN_CLKSOURCE_PLL2

PLL2 P clock used as FDCAN kernel clock source

EXTERNAL HSE Mode

LL_RCC_HSE_ANALOG_MODE

HSE clock used as ANALOG clock source

LL_RCC_HSE_DIGITAL_MODE

HSE clock used as DIGITAL clock source

Peripheral HSPI get clock source

LL_RCC_HSPI_CLKSOURCE

HSPI Clock source selection

Peripheral HSPI1 kernel clock source selection

LL_RCC_HSPI_CLKSOURCE_SYSCLK

LL_RCC_HSPI_CLKSOURCE_PLL1

LL_RCC_HSPI_CLKSOURCE_PLL2

LL_RCC_HSPI_CLKSOURCE_PLL3

Peripheral I2Cx get clock source

LL_RCC_I2C1_CLKSOURCE

I2C1 Clock source selection

LL_RCC_I2C2_CLKSOURCE

I2C2 Clock source selection

LL_RCC_I2C3_CLKSOURCE

I2C3 Clock source selection

LL_RCC_I2C4_CLKSOURCE

I2C4 Clock source selection

LL_RCC_I2C5_CLKSOURCE

I2C1 Clock source selection

LL_RCC_I2C6_CLKSOURCE

I2C1 Clock source selection

Peripheral I2Cx clock source selection

LL_RCC_I2C1_CLKSOURCE_PCLK1

PCLK1 clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_HSI

HSI clock used as I2C1 clock source

LL_RCC_I2C1_CLKSOURCE_MSIK

MSIK clock used as I2C1 clock source

LL_RCC_I2C2_CLKSOURCE_PCLK1

PCLK1 clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_HSI

HSI clock used as I2C2 clock source

LL_RCC_I2C2_CLKSOURCE_MSIK

MSIK clock used as I2C2 clock source

LL_RCC_I2C3_CLKSOURCE_PCLK3

PCLK3 clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_HSI

HSI clock used as I2C3 clock source

LL_RCC_I2C3_CLKSOURCE_MSIK

MSIK clock used as I2C3 clock source

LL_RCC_I2C4_CLKSOURCE_PCLK1

PCLK1 clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_HSI

HSI clock used as I2C4 clock source

LL_RCC_I2C4_CLKSOURCE_MSIK

MSIK clock used as I2C4 clock source

LL_RCC_I2C5_CLKSOURCE_PCLK1

PCLK1 clock used as I2C5 clock source

LL_RCC_I2C5_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C5 clock source

LL_RCC_I2C5_CLKSOURCE_HSI

HSI clock used as I2C5 clock source

LL_RCC_I2C5_CLKSOURCE_MSIK

MSIK clock used as I2C5 clock source

LL_RCC_I2C6_CLKSOURCE_PCLK1

PCLK1 clock used as I2C6 clock source

LL_RCC_I2C6_CLKSOURCE_SYSCLK

SYSCLK clock used as I2C6 clock source

LL_RCC_I2C6_CLKSOURCE_HSI

HSI clock used as I2C6 clock source

LL_RCC_I2C6_CLKSOURCE_MSIK

MSIK clock used as I2C6 clock source

Peripheral LPTIMx get clock source

LL_RCC_LPTIM1_CLKSOURCE

LPTIM1 Clock source selection

LL_RCC_LPTIM2_CLKSOURCE

LPTIM2 Clock source selection

LL_RCC_LPTIM34_CLKSOURCE

LPTIM3 and LPTIM4 Clock source selection

Peripheral LPTIMx clock source selection

LL_RCC_LPTIM1_CLKSOURCE_MSIK

MSIK clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_LSI

LSI clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_HSI

HSI clock used as LPTIM1 clock source

LL_RCC_LPTIM1_CLKSOURCE_LSE

LSE clock used as LPTIM1 clock source

LL_RCC_LPTIM2_CLKSOURCE_PCLK1

PCLK1 clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_LSI

LSI clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_HSI

HSI clock used as LPTIM2 clock source

LL_RCC_LPTIM2_CLKSOURCE_LSE

LSE clock used as LPTIM2 clock source

LL_RCC_LPTIM34_CLKSOURCE_MSIK

MSIK clock used as LPTIM34 clock source

LL_RCC_LPTIM34_CLKSOURCE_LSI

LSI clock used as LPTIM34 clock source

LL_RCC_LPTIM34_CLKSOURCE_HSI

HSI clock used as LPTIM34 clock source

LL_RCC_LPTIM34_CLKSOURCE_LSE

LSE clock used as LPTIM34 clock source

Peripheral LPUARTx get clock source

LL_RCC_LPUART1_CLKSOURCE

LPUART1 Clock source selection

Peripheral LPUARTx clock source selection

LL_RCC_LPUART1_CLKSOURCE_PCLK3

PCLK3 clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_SYSCLK

SYSCLK clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_HSI

HSI clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_LSE

LSE clock used as LPUART1 clock source

LL_RCC_LPUART1_CLKSOURCE_MSIK

MSIK clock used as LPUART1 clock source

LSCO Selection

LL_RCC_LSCO_CLKSOURCE_LSI

LSI selection for low speed clock

LL_RCC_LSCO_CLKSOURCE_LSE

LSE selection for low speed clock

LSE oscillator drive capability

LL_RCC_LSEDRIVE_LOW

Xtal mode lower driving capability

LL_RCC_LSEDRIVE_MEDIUMLOW

Xtal mode medium low driving capability

LL_RCC_LSEDRIVE_MEDIUMHIGH

Xtal mode medium high driving capability

LL_RCC_LSEDRIVE_HIGH

Xtal mode higher driving capability

LSI prescaler

LL_RCC_LSI_DIV_1

LSI divided by 1

LL_RCC_LSI_DIV_128

LSI divided by 128

Peripheral LTDC get clock source

LL_RCC_LTDC_CLKSOURCE

Peripheral LTDC clock source selection

LL_RCC_LTDC_CLKSOURCE_PLL2

LL_RCC_LTDC_CLKSOURCE_PLL3

MCO1 SOURCE selection

LL_RCC_MCO1SOURCE_NOCLOCK

MCO output disabled, no clock on MCO

LL_RCC_MCO1SOURCE_SYSCLK

SYSCLK selection as MCO1 source

LL_RCC_MCO1SOURCE_MSIS

MSIS selection as MCO1 source

LL_RCC_MCO1SOURCE_HSI

HSI selection as MCO1 source

LL_RCC_MCO1SOURCE_HSE

HSE selection as MCO1 source

LL_RCC_MCO1SOURCE_PLLCLK

Main PLL selection as MCO1 source

LL_RCC_MCO1SOURCE_LSI

LSI selection as MCO1 source

LL_RCC_MCO1SOURCE_LSE

LSE selection as MCO1 source

LL_RCC_MCO1SOURCE_HSI48

HSI48 selection as MCO1 source

LL_RCC_MCO1SOURCE_MSIK

MSIK selection as MCO1 source

MCO1 prescaler

LL_RCC_MCO1_DIV_1

MCO not divided

LL_RCC_MCO1_DIV_2

MCO divided by 2

LL_RCC_MCO1_DIV_4

MCO divided by 4

LL_RCC_MCO1_DIV_8

MCO divided by 8

LL_RCC_MCO1_DIV_16

MCO divided by 16

Peripheral MDF1 get clock source

LL_RCC_MDF1_CLKSOURCE

Peripheral MDF1 clock source selection

LL_RCC_MDF1_CLKSOURCE_HCLK

HCLK clock used as MDF1 clock

LL_RCC_MDF1_CLKSOURCE_PLL1

PLL1 clock used as MDF1 clock

LL_RCC_MDF1_CLKSOURCE_PLL3

PLL3 clock used as MDF1 clock

LL_RCC_MDF1_CLKSOURCE_MSIK

MSIK clock used as MDF1 clock

LL_RCC_MDF1_CLKSOURCE_PIN

PIN SAI1_EXTCLK clock used as MDF1 clock

MSI BIAS Mode

LL_RCC_MSIBIASMODE_SAMPLING

Sampling mode selection for MSI

LL_RCC_MSIBIASMODE_CONTINUOUS

Continuous mode selection for MSI

MSIK Clock Range

LL_RCC_MSIKRANGE_0

MSIK = 48 MHz

LL_RCC_MSIKRANGE_1

MSIK = 24 MHz

LL_RCC_MSIKRANGE_2

MSIK = 16 MHz

LL_RCC_MSIKRANGE_3

MSIK = 12 MHz

LL_RCC_MSIKRANGE_4
MSIK = 4 MHz

LL_RCC_MSIKRANGE_5
MSIK = 2 MHz

LL_RCC_MSIKRANGE_6
MSIK = 1.5 MHz

LL_RCC_MSIKRANGE_7
MSIK = 1 MHz

LL_RCC_MSIKRANGE_8
MSIK = 3.072 MHz

LL_RCC_MSIKRANGE_9
MSIK = 1.536 MHz

LL_RCC_MSIKRANGE_10
MSIK = 1.024 MHz

LL_RCC_MSIKRANGE_11
MSIK = 768 KHz

LL_RCC_MSIKRANGE_12
MSIK = 400 KHz

LL_RCC_MSIKRANGE_13
MSIK = 200 KHz

LL_RCC_MSIKRANGE_14
MSIK = 150 KHz

LL_RCC_MSIKRANGE_15
MSIK = 100 KHz

MSIK range after Standby mode

LL_RCC_MSIKSRANGE_4
MSIK = 4 MHz

LL_RCC_MSIKSRANGE_5
MSIK = 2 MHz

LL_RCC_MSIKSRANGE_6
MSIK = 1.5 MHz

LL_RCC_MSIKSRANGE_7
MSIK = 1 MHz

LL_RCC_MSIKSRANGE_8
MSIK = 3.072 MHz

MSIS/MSIK Pll Mode

LL_RCC_PLLMODE_MSIS
MSIS selection for Pll Mode

LL_RCC_PLLMODE_MSIK

MSIK selection for Pll Mode

MSI clock range selection

LL_RCC_MSIRANGESEL_STANDBY

MSI Range is provided by MSISRANGE

LL_RCC_MSIRANGESEL_RUN

MSI Range is provided by MSISRANGE

MSIS Clock Range

LL_RCC_MSISRANGE_0

MSIS = 48 MHz

LL_RCC_MSISRANGE_1

MSIS = 24 MHz

LL_RCC_MSISRANGE_2

MSIS = 16 MHz

LL_RCC_MSISRANGE_3

MSIS = 12 MHz

LL_RCC_MSISRANGE_4

MSIS = 4 MHz

LL_RCC_MSISRANGE_5

MSIS = 2 MHz

LL_RCC_MSISRANGE_6

MSIS = 1.5 MHz

LL_RCC_MSISRANGE_7

MSIS = 1 MHz

LL_RCC_MSISRANGE_8

MSIS = 3.072 MHz

LL_RCC_MSISRANGE_9

MSIS = 1.536 MHz

LL_RCC_MSISRANGE_10

MSIS = 1.024 MHz

LL_RCC_MSISRANGE_11

MSIS = 768 KHz

LL_RCC_MSISRANGE_12

MSIS = 400 KHz

LL_RCC_MSISRANGE_13

MSIS = 200 KHz

LL_RCC_MSISRANGE_14

MSIS = 150 KHz

LL_RCC_MSISRANGE_15

MSIS = 100 KHz

MSIS range after Standby mode

LL_RCC_MSISSRANGE_4

MSIS = 4 MHz

LL_RCC_MSISSRANGE_5

MSIS = 2 MHz

LL_RCC_MSISSRANGE_6

MSIS = 1.5 MHz

LL_RCC_MSISSRANGE_7

MSIS = 1 MHz

LL_RCC_MSISSRANGE_8

MSIS = 3.072 MHz

MSI clock Trimming

LL_RCC_MSI_OSCILLATOR_0

MSI clock trimming for ranges 0 to 3

LL_RCC_MSI_OSCILLATOR_1

MSI clock trimming for ranges 4 to 7

LL_RCC_MSI_OSCILLATOR_2

MSI clock trimming for ranges 8 to 11

LL_RCC_MSI_OSCILLATOR_3

MSI clock trimming for ranges 12 to 15

Peripheral OCTOSPI get clock source

LL_RCC_OCTOSPI_CLKSOURCE

OctoSPI Clock source selection

Peripheral OCTOSPI kernel clock source selection

LL_RCC_OCTOSPI_CLKSOURCE_SYSCLK

SYSCLK clock used as OctoSPI kernel clock source

LL_RCC_OCTOSPI_CLKSOURCE_MSIK

MSIK clock used as OctoSPI kernel clock source

LL_RCC_OCTOSPI_CLKSOURCE_PLL1

PLL1 "Q" clock used as OctoSPI kernel clock source

LL_RCC_OCTOSPI_CLKSOURCE_PLL2

PLL2 "Q" clock used as OctoSPI kernel clock source

Oscillator Values adaptation

EXTERNAL_SAI2_CLOCK_VALUE

Value of the SAI2_EXTCLK external oscillator in Hz

Peripheral clock frequency

LL_RCC_PERIPH_FREQUENCY_NO

No clock enabled for the peripheral

LL_RCC_PERIPH_FREQUENCY_NA

Frequency cannot be provided as external clock

EPOD prescaler

LL_RCC_PLL1MBOOST_DIV_1

PLL1CLK not divided

LL_RCC_PLL1MBOOST_DIV_2

PLL1CLK divided by 2

LL_RCC_PLL1MBOOST_DIV_4

PLL1CLK divided by 4

LL_RCC_PLL1MBOOST_DIV_6

PLL1CLK divided by 6

LL_RCC_PLL1MBOOST_DIV_8

PLL1CLK divided by 8

LL_RCC_PLL1MBOOST_DIV_10

PLL1CLK divided by 10

LL_RCC_PLL1MBOOST_DIV_12

PLL1CLK divided by 12

LL_RCC_PLL1MBOOST_DIV_14

PLL1CLK divided by 14

LL_RCC_PLL1MBOOST_DIV_16

PLL1CLK divided by 16

PLL1 entry clock source

LL_RCC_PLL1SOURCE_NONE

No clock selected as main PLL1 entry clock source

LL_RCC_PLL1SOURCE_MSIS

MSIS clock selected as main PLL1 entry clock source

LL_RCC_PLL1SOURCE_HSI

HSI clock selected as main PLL1 entry clock source

LL_RCC_PLL1SOURCE_HSE

HSE clock selected as main PLL1 entry clock source

LL_RCC_PLLSOURCE_NONE

alias define for compatibility with legacy code

LL_RCC_PLLSOURCE_MSIS

alias define for compatibility with legacy code

LL_RCC_PLLSOURCE_HSI

alias define for compatibility with legacy code

LL_RCC_PLLSOURCE_HSE

alias define for compatibility with legacy code

PLL2 entry clock source

LL_RCC_PLL2SOURCE_NONE

No clock selected as main PLL2 entry clock source

LL_RCC_PLL2SOURCE_MSIS

MSIS clock selected as main PLL2 entry clock source

LL_RCC_PLL2SOURCE_HSI

HSI clock selected as main PLL2 entry clock source

LL_RCC_PLL2SOURCE_HSE

HSE clock selected as main PLL2 entry clock source

PLL3 entry clock source

LL_RCC_PLL3SOURCE_NONE

No clock selected as main PLL3 entry clock source

LL_RCC_PLL3SOURCE_MSIS

MSIS clock selected as main PLL3 entry clock source

LL_RCC_PLL3SOURCE_HSI

HSI clock selected as main PLL3 entry clock source

LL_RCC_PLL3SOURCE_HSE

HSE clock selected as main PLL3 entry clock source

All PLLs input ranges

LL_RCC_PLLINPUTRANGE_4_8

VCO input range: 4 to 8 MHz

LL_RCC_PLLINPUTRANGE_8_16

VCO input range: 8 to 16 MHz

Peripheral RNG get clock source

LL_RCC_RNG_CLKSOURCE

RNG Clock source selection

Peripheral RNG clock source selection

LL_RCC_RNG_CLKSOURCE_HSI48

HSI48 clock used as RNG clock source

LL_RCC_RNG_CLKSOURCE_HSI48_DIV2

HSI48/2 clock used as RNG clock source

LL_RCC_RNG_CLKSOURCE_HSI

HSI clock used as RNG clock source

RTC clock source selection

LL_RCC_RTC_CLKSOURCE_NONE

No clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_LSE

LSE oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_LSI

LSI oscillator clock used as RTC clock

LL_RCC_RTC_CLKSOURCE_HSE_DIV32

HSE oscillator clock divided by 32 used as RTC clock

Peripheral SAES get clock source

LL_RCC_SAES_CLKSOURCE

SAES Clock source selection

Peripheral SAES clock source selection

LL_RCC_SAES_CLKSOURCE_SHSI

SHSI clock used as SAES clock source

LL_RCC_SAES_CLKSOURCE_SHSI_DIV2

SHSI_DIV2 clock used as SAES clock source

Peripheral SAIx get clock source

LL_RCC_SAI1_CLKSOURCE

SAI1 Clock source selection

LL_RCC_SAI2_CLKSOURCE

SAI2 Clock source selection

Peripheral SAIx clock source selection

LL_RCC_SAI1_CLKSOURCE_PLL2

PLL2 clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PLL3

PLL3 clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PLL1

PLL1 clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_HSI

HSI clock used as SAI1 clock source

LL_RCC_SAI1_CLKSOURCE_PIN

External input clock used as SAI1 clock source

LL_RCC_SAI2_CLKSOURCE_PLL2

PLL2 clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_PLL3

PLL3 clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_PLL1
PLL1clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_HSI
HSI clock used as SAI2 clock source

LL_RCC_SAI2_CLKSOURCE_PIN
External input clock used as SAI2 clock source

/2 Peripheral SDMMC get clock source

LL_RCC_SDMMC_CLKSOURCE
SDMMC1/2 Clock source selection

Peripheral SDMMC clock source selection

LL_RCC_SDMMC12_CLKSOURCE_HSI48
HSI48 clock used as SDMMC1/2 clock source

LL_RCC_SDMMC12_CLKSOURCE_PLL2
PLL2 "Q" clock used as SDMMC1/2 clock source

LL_RCC_SDMMC12_CLKSOURCE_PLL1
PLL1 "Q" clock used as SDMMC1/2 clock source

LL_RCC_SDMMC12_CLKSOURCE_MSIK
MSIK clock used as SDMMC1/2 clock source

Peripheral SDMMC get kernel clock source

LL_RCC_SDMMC_KERNELCLKSOURCE
SDMMC1/2 Kernel Clock source selection

Peripheral SDMMC1/2 kernel clock source selection

LL_RCC_SDMMC12_KERNELCLKSOURCE_48CLK
48MHz clock from internal multiplexor used as SDMMC1/2 clock source

LL_RCC_SDMMC12_KERNELCLKSOURCE_PLL1
PLL1 "P" used as SDMMC1/2 clock source

Peripheral SPIx get clock source

LL_RCC_SPI1_CLKSOURCE
SPI1 Clock source selection

LL_RCC_SPI2_CLKSOURCE
SPI2 Clock source selection

LL_RCC_SPI3_CLKSOURCE
SPI3 Clock source selection

Peripheral SPIx clock source selection

LL_RCC_SPI1_CLKSOURCE_PCLK2
PCLK2 clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_SYSCLK
SYSCLK clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_HSI
HSI clock used as SPI1 clock source

LL_RCC_SPI1_CLKSOURCE_MSIK
MSIK clock used as SPI1 clock source

LL_RCC_SPI2_CLKSOURCE_PCLK1
PCLK1 clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_SYSCLK
SYSCLK clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_HSI
HSI clock used as SPI2 clock source

LL_RCC_SPI2_CLKSOURCE_MSIK
MSIK clock used as SPI2 clock source

LL_RCC_SPI3_CLKSOURCE_PCLK3
PCLK3 clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_SYSCLK
SYSCLK clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_HSI
HSI clock used as SPI3 clock source

LL_RCC_SPI3_CLKSOURCE_MSIK
MSIK clock used as SPI3 clock source

Wakeup from Stop and CSS backup clock selection

LL_RCC_STOP_WAKEUPCLOCK_MSIS
MSIS selection after wake-up from STOP

LL_RCC_STOP_WAKEUPCLOCK_HSI
HSI selection after wake-up from STOP

Wakeup from Stop kernel clock automatic enable selection

LL_RCC_STOP_WAKEUPKERCLOCK_MSIK
MSIK oscillator automatically enabled when exiting Stop mode

LL_RCC_STOP_WAKEUPKERCLOCK_HSI
HSI oscillator automatically enabled when exiting Stop mode

AHB prescaler

LL_RCC_SYSCLK_DIV_1
SYSCLK not divided

LL_RCC_SYSCLK_DIV_2
SYSCLK divided by 2

LL_RCC_SYSCLK_DIV_4
SYSCLK divided by 4

LL_RCC_SYSCLK_DIV_8
SYSCLK divided by 8

LL_RCC_SYSCLK_DIV_16
SYSCLK divided by 16

LL_RCC_SYSCLK_DIV_64
SYSCLK divided by 64

LL_RCC_SYSCLK_DIV_128
SYSCLK divided by 128

LL_RCC_SYSCLK_DIV_256
SYSCLK divided by 256

LL_RCC_SYSCLK_DIV_512
SYSCLK divided by 512

SYSTICK clock source selection

LL_RCC_SYSTICK_CLKSOURCE_HCLKDIV8
HCLKDIV8 clock used as SYSTICK clock source

LL_RCC_SYSTICK_CLKSOURCE_LSI
LSI clock used as SYSTICK clock source

LL_RCC_SYSTICK_CLKSOURCE_LSE
LSE clock used as SYSTICK clock source

System clock switch

LL_RCC_SYS_CLKSOURCE_MSIS
MSIS selection as system clock

LL_RCC_SYS_CLKSOURCE_HSI
HSI oscillator selection as system clock

LL_RCC_SYS_CLKSOURCE_HSE
HSE selection as system clock

LL_RCC_SYS_CLKSOURCE_PLL1
PLL selection as system clock

System clock switch status

LL_RCC_SYS_CLKSOURCE_STATUS_MSIS
MSIS used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_HSI
HSI used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_HSE
HSE used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_PLL1

PLL1 used as system clock

TIM Input capture clock source selection

LL_RCC_TIMIC_CLKSOURCE_NONE

No clock available for TIM16/TIM17 and LPTIM2 input capture

LL_RCC_TIMIC_CLKSOURCE_HSI_DIV256

HSI/256 selected for TIM16/TIM17 and LPTIM2 input capture

LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV1024

MSIS/1024 selected for TIM16/TIM17 and LPTIM2 input capture

LL_RCC_TIMIC_CLKSOURCE_MSIS_DIV4

MSIS/4 selected for TIM16/TIM17 and LPTIM2 input capture

LL_RCC_TIMIC_CLKSOURCE_MSIK_DIV4

MSIK/4 selected for TIM16/TIM17 and LPTIM2 input capture

LL_RCC_TIMIC_CLKSOURCE_MSIK_DIV1024

MSIK/1024 selected for TIM16/TIM17 and LPTIM2 input capture

Peripheral UARTx get clock source

LL_RCC_UART4_CLKSOURCE

UART4 Clock source selection

LL_RCC_UART5_CLKSOURCE

UART5 Clock source selection

Peripheral UARTx clock source selection

LL_RCC_UART4_CLKSOURCE_PCLK1

PCLK1 clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_SYSCLK

SYSCLK clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_HSI

HSI clock used as UART4 clock source

LL_RCC_UART4_CLKSOURCE_LSE

LSE clock used as UART4 clock source

LL_RCC_UART5_CLKSOURCE_PCLK1

PCLK1 clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_SYSCLK

SYSCLK clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_HSI

HSI clock used as UART5 clock source

LL_RCC_UART5_CLKSOURCE_LSE

LSE clock used as UART5 clock source

Peripheral USARTx get clock source

LL_RCC_USART1_CLKSOURCE
USART1 Clock source selection

LL_RCC_USART2_CLKSOURCE
USART2 Clock source selection

LL_RCC_USART3_CLKSOURCE
USART3 Clock source selection

LL_RCC_USART6_CLKSOURCE
USART6 Clock source selection

Peripheral USARTx clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK2
PCLK2 clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_SYSCLK
SYSCLK clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_HSI
HSI clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_LSE
LSE clock used as USART1 clock source

LL_RCC_USART2_CLKSOURCE_PCLK1
PCLK1 clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_SYSCLK
SYSCLK clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_HSI
HSI clock used as USART2 clock source

LL_RCC_USART2_CLKSOURCE_LSE
LSE clock used as USART2 clock source

LL_RCC_USART3_CLKSOURCE_PCLK1
PCLK3 clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_SYSCLK
SYSCLK clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_HSI
HSI clock used as USART3 clock source

LL_RCC_USART3_CLKSOURCE_LSE
LSE clock used as USART3 clock source

LL_RCC_USART6_CLKSOURCE_PCLK1
PCLK1 clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_SYSCLK
SYSCLK clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_HSI
HSI clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_LSE

LSE clock used as USART6 clock source

LL_RCC_USART6_CLKSOURCE_PCLK2

Peripheral USB get clock source

LL_RCC_USB_CLKSOURCE

USB Clock source selection

Peripheral USBPHY get clock source

LL_RCC_USBPHY_CLKSOURCE

Peripheral USBPHY clock source selection

LL_RCC_USBPHYCLKSOURCE_HSE

HSE clock selected as USBPHYC clock

LL_RCC_USBPHYCLKSOURCE_HSE_DIV2

HSE clock divided by 2 selected as USBPHYC clock

LL_RCC_USBPHYCLKSOURCE_PLL1

PLL1 divider P selected as USBPHYC clock

LL_RCC_USBPHYCLKSOURCE_PLL1_DIV2

PLL1 divider P divided by 2 selected as USBPHYC clock

Peripheral USB clock source selection

LL_RCC_USB_CLKSOURCE_HSI48

HSI48 clock used as USB clock source

LL_RCC_USB_CLKSOURCE_PLL2

PLL2 "Q" clock used as USB clock source

LL_RCC_USB_CLKSOURCE_PLL1

PLL1 "Q" clock used as USB clock source

LL_RCC_USB_CLKSOURCE_MSIK

MSIK clock used as USB clock source

MSI

LL_RCC_MSI_Enable

alias define for compatibility with legacy code

LL_RCC_MSI_Disable

alias define for compatibility with legacy code

LL_RCC_MSI_IsReady

alias define for compatibility with legacy code

LL_RCC_MSI_SetRange

alias define for compatibility with legacy code

LL_RCC_MSI_GetRange

alias define for compatibility with legacy code

LL_RCC_MSI_SetRangeAfterStandby
alias define for compatibility with legacy code

LL_RCC_MSI_GetRangeAfterStandby
alias define for compatibility with legacy code

PLL1

LL_RCC_PLL_Enable
alias for compatibility with legacy code

LL_RCC_PLL_Disable
alias for compatibility with legacy code

LL_RCC_PLL_IsReady
alias for compatibility with legacy code

LL_RCC_PLL_ConfigDomain_SYS
alias for compatibility with legacy code

LL_RCC_PLL_ConfigDomain_SAI
alias for compatibility with legacy code

LL_RCC_PLL_ConfigDomain_48M
alias for compatibility with legacy code

LL_RCC_PLL_SetMainSource
alias for compatibility with legacy code

LL_RCC_PLL_GetMainSource
alias for compatibility with legacy code

LL_RCC_PLL_SetN
alias for compatibility with legacy code

LL_RCC_PLL_GetN
alias for compatibility with legacy code

LL_RCC_PLL_SetP
alias for compatibility with legacy code

LL_RCC_PLL_GetP
alias for compatibility with legacy code

LL_RCC_PLL_SetQ
alias for compatibility with legacy code

LL_RCC_PLL_GetQ
alias for compatibility with legacy code

LL_RCC_PLL_SetR
alias for compatibility with legacy code

LL_RCC_PLL_GetR
alias for compatibility with legacy code

LL_RCC_PLL_SetDivider
alias for compatibility with legacy code

LL_RCC_PLL_GetDivider
alias for compatibility with legacy code

LL_RCC_PLL_EnableDomain_SAI
alias for compatibility with legacy code

LL_RCC_PLL_DisableDomain_SAI
alias for compatibility with legacy code

LL_RCC_PLL_IsEnabledDomain_SAI
alias for compatibility with legacy code

LL_RCC_PLL_DisableDomain_48M
alias for compatibility with legacy code

LL_RCC_PLL_IsEnabledDomain_48M
alias for compatibility with legacy code

LL_RCC_PLL_EnableDomain_SYS
alias for compatibility with legacy code

LL_RCC_PLL_DisableDomain_SYS
alias for compatibility with legacy code

LL_RCC_PLL_IsEnabledDomain_SYS
alias for compatibility with legacy code

LL_RCC_PLLFRACN_Enable
alias for compatibility with legacy code

LL_RCC_PLLFRACN_IsEnabled
alias for compatibility with legacy code

LL_RCC_PLLFRACN_Disable
alias for compatibility with legacy code

LL_RCC_PLL_SetFRACN
alias for compatibility with legacy code

LL_RCC_PLL_GetFRACN
alias for compatibility with legacy code

LL_RCC_PLL_SetVCOInputRange
alias for compatibility with legacy code

Security Services

LL_RCC_ALL_NSEC
No security on RCC resources (default)

LL_RCC_ALL_SEC
Security on all RCC resources

LL_RCC_HSI_SEC
HSI clock configuration security

LL_RCC_HSI_NSEC
HSI clock configuration secure/non-secure access

LL_RCC_HSE_SEC
HSE clock configuration security

LL_RCC_HSE_NSEC
HSE clock configuration secure/non-secure access

LL_RCC_MSI_SEC
MSI clock configuration security

LL_RCC_MSI_NSEC
MSI clock configuration secure/non-secure access

LL_RCC_LSE_SEC
LSE clock configuration security

LL_RCC_LSE_NSEC
LSE clock configuration secure/non-secure access

LL_RCC_LSI_SEC
LSI clock configuration security

LL_RCC_LSI_NSEC
LSI clock configuration secure/non-secure access

LL_RCC_SYSCLK_SEC
SYSCLK clock; STOPWUCK and MCO output configuration security

LL_RCC_SYSCLK_NSEC
SYSCLK clock; STOPWUCK and MCO output configuration secure/non-secure access

LL_RCC_PRESCALERS_SEC
AHBx/APBx prescaler configuration security

LL_RCC_PRESCALERS_NSEC
AHBx/APBx prescaler configuration secure/non-secure access

LL_RCC_PLL1_SEC
PLL1 clock configuration security

LL_RCC_PLL1_NSEC
main PLL1 clock configuration secure/non-secure access

LL_RCC_PLL2_SEC
PLL2 clock configuration security

LL_RCC_PLL2_NSEC
main PLL2 clock configuration secure/non-secure access

LL_RCC_PLL3_SEC
PLL3 clock configuration security

LL_RCC_PLL3_NSEC
main PLL3 clock configuration secure/non-secure access

LL_RCC_ICLK_SEC
ICLK clock source selection security

LL_RCC_ICLK_NSEC

ICLK clock source selection secure/non-secure access

LL_RCC_HSI48_SEC

HSI48 clock configuration security

LL_RCC_HSI48_NSEC

HSI48 clock configuration secure/non-secure access

LL_RCC_RESET_FLAGS_SEC

Remove reset flag security

LL_RCC_RESET_FLAGS_NSEC

LL_RCC_PLL_SEC

alias define for compatibility with legacy code

LL_RCC_PLL_NSEC

alias define for compatibility with legacy code

LL_RCC_CLK48M_SEC

alias define for compatibility with legacy code

LL_RCC_CLK48M_NSEC

alias define for compatibility with legacy code

Calculate frequencies

__LL_RCC_CALC_PLL1CLK_FREQ

Description:
- Helper macro to calculate the PLL1CLK frequency on system domain.

Parameters:
- __INPUTFREQ__: PLL1 Input frequency (based on MSI/HSE/HSI)
- __PLL1M__: parameter can be a value between 1 and 16
- __PLL1N__: parameter can be a value between 4 and 512
- __PLL1R__: parameter can be a value between 1 and 128 (Only division by 1 and even division are allowed)

Return value:
- PLL1: clock frequency (in Hz)

Notes:
- ex: __LL_RCC_CALC_PLL1CLK_FREQ (HSE_VALUE,LL_RCC_PLL1_GetDivider (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetR ());

__LL_RCC_CALC_PLLCLK_FREQ

alias for compatibility with legacy code

__LL_RCC_CALC_PLL1CLK_SAI_FREQ

Description:

- Helper macro to calculate the PLL1CLK frequency used on SAI domain.

Parameters:

- __INPUTFREQ__: PLL1 Input frequency (based on MSI/HSE/HSI)
- __PLL1M__: parameter can be a value between 1 and 16
- __PLL1N__: parameter can be a value between 4 and 512
- __PLL1P__: parameter can be a value between 1 and 128

Return value:

- PLL1: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL1CLK_SAI_FREQ (HSE_VALUE,LL_RCC_PLL1_GetDivider (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetP ());

__LL_RCC_CALC_PLLCLK_SAI_FREQ

alias for compatibility with legacy code

__LL_RCC_CALC_PLL1CLK_48M_FREQ

Description:

- Helper macro to calculate the PLL1CLK frequency used on 48M domain.

Parameters:

- __INPUTFREQ__: PLL1 Input frequency (based on MSI/HSE/HSI)
- __PLL1M__: parameter can be a value between 1 and 16
- __PLL1N__: parameter can be a value between 4 and 512
- __PLL1Q__: parameter can be a value between 1 and 128

Return value:

- PLL: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL1CLK_48M_FREQ (HSE_VALUE,LL_RCC_PLL1_GetDivider (), LL_RCC_PLL1_GetN (), LL_RCC_PLL1_GetQ ());

__LL_RCC_CALC_PLLCLK_48M_FREQ

alias for compatibility with legacy code

__LL_RCC_CALC_PLL2CLK_SAI_FREQ

Description:

- Helper macro to calculate the PLL2 frequency used for SAI domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLL2M__: parameter can be a value between 1 and 16
- __PLL2N__: parameter can be a value between 4 and 512
- __PLL2P__: parameter can be a value between 1 and 128

Return value:

- PLL2: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL2CLK_SAI_FREQ (HSE_ALUE,LL_RCC_PLL2_GetDivider (), LL_RCC_PLL2_GetN (), LL_RCC_PLL2_GetP ());

__LL_RCC_CALC_PLL2CLK_48M_FREQ

Description:

- Helper macro to calculate the PLL2 frequency used on 48M domain.

Parameters:

- __INPUTFREQ__: PLL Input frequency (based on MSI/HSE/HSI)
- __PLL2M__: parameter can be a value between 1 and 16
- __PLL2N__: parameter can be a value between 4 and 512
- __PLL2Q__: parameter can be a value between 1 and 128

Return value:

- PLL2: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL2CLK_48M_FREQ (HSE_VALUE,LL_RCC_PLL2_GetDivider (), LL_RCC_PLL2_GetN (), LL_RCC_PLL2_GetQ ());

__LL_RCC_CALC_PLL2CLK_ADC_FREQ

Description:

- Helper macro to calculate the PLL2 frequency used on ADC domain.

Parameters:

- __INPUTFREQ__: PLL2 Input frequency (based on MSI/HSE/HSI)
- __PLL2M__: parameter can be a value between 1 and 16
- __PLL2N__: parameter can be a value between 4 and 512
- __PLL2R__: parameter can be a value between 1 and 128

Return value:

- PLL2: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL2CLK_ADC_FREQ (HSE_VALUE,LL_RCC_PLL2_GetDivider (), LL_RCC_PLL2_GetN (), LL_RCC_PLL2_GetR ());

__LL_RCC_CALC_PLL3CLK_SAI_FREQ

Description:

- Helper macro to calculate the PLL3 frequency used for SAI domain.

Parameters:

- __INPUTFREQ__: PLL3 Input frequency (based on MSI/HSE/HSI)
- __PLL3M__: parameter can be a value between 1 and 16
- __PLL3N__: parameter can be a value between 4 and 512
- __PLL3P__: parameter can be a value between 1 and 128

Return value:

- PLL3: clock frequency (in Hz)

Notes:

- ex: __LL_RCC_CALC_PLL3CLK_SAI_FREQ (HSE_VALUE,LL_RCC_PLL3_GetDivider (), LL_RCC_PLL3_GetN (), LL_RCC_PLL3_GetP ());

__LL_RCC_CALC_PLL3CLK_48M_FREQ

Description:
- Helper macro to calculate the PLL2 frequency used on 48M domain.

Parameters:
- __INPUTFREQ__: PLL3 Input frequency (based on MSI/HSE/HSI)
- __PLL3M__: parameter can be a value between 1 and 16
- __PLL3N__: parameter can be a value between 4 and 512
- __PLL3Q__: parameter can be a value between 1 and 128

Return value:
- PLL3: clock frequency (in Hz)

Notes:
- ex: __LL_RCC_CALC_PLL3CLK_48M_FREQ (HSE_VALUE,LL_RCC_PLL3_GetDivider (), LL_RCC_PLL3_GetN (), LL_RCC_PLL3_GetQ ());

__LL_RCC_CALC_PLL3CLK_HSPI_LTDC_FREQ

Description:
- Helper macro to calculate the PLL3 frequency used on HSPI domain.

Parameters:
- __INPUTFREQ__: PLL3 Input frequency (based on MSI/HSE/HSI)
- __PLL3M__: parameter can be a value between 1 and 16
- __PLL3N__: parameter can be a value between 4 and 512
- __PLL3R__: parameter can be a value between 1 and 128

Return value:
- PLL3: clock frequency (in Hz)

Notes:
- ex: __LL_RCC_CALC_PLL3CLK_HSPI_LTDC_FREQ (HSE_VALUE,LL_RCC_PLL3_GetDivider (), LL_RCC_PLL3_GetN (), LL_RCC_PLL3_GetQ ());

__LL_RCC_CALC_PLL3CLK_HSPI_FREQ

__LL_RCC_CALC_HCLK_FREQ

Description:
- Helper macro to calculate the HCLK frequency.

Parameters:
- __SYSCLKFREQ__: SYSCLK frequency (based on MSI/HSE/HSI/PLLCLK)
- __AHBPRESCALER__: This parameter can be one of the following values:
  – LL_RCC_SYSCLK_DIV_1
  – LL_RCC_SYSCLK_DIV_2
  – LL_RCC_SYSCLK_DIV_4
  – LL_RCC_SYSCLK_DIV_8
  – LL_RCC_SYSCLK_DIV_16
  – LL_RCC_SYSCLK_DIV_64
  – LL_RCC_SYSCLK_DIV_128
  – LL_RCC_SYSCLK_DIV_256
  – LL_RCC_SYSCLK_DIV_512

Return value:
- HCLK: clock frequency (in Hz)

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (ABP1)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB1PRESCALER__: This parameter can be one of the following values:
  – LL_RCC_APB1_DIV_1
  – LL_RCC_APB1_DIV_2
  – LL_RCC_APB1_DIV_4
  – LL_RCC_APB1_DIV_8
  – LL_RCC_APB1_DIV_16

Return value:

- PCLK1: clock frequency (in Hz)

__LL_RCC_CALC_PCLK2_FREQ

Description:

- Helper macro to calculate the PCLK2 frequency (ABP2)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB2PRESCALER__: This parameter can be one of the following values:
  – LL_RCC_APB2_DIV_1
  – LL_RCC_APB2_DIV_2
  – LL_RCC_APB2_DIV_4
  – LL_RCC_APB2_DIV_8
  – LL_RCC_APB2_DIV_16

Return value:

- PCLK2: clock frequency (in Hz)

__LL_RCC_CALC_PCLK3_FREQ

Description:

- Helper macro to calculate the PCLK3 frequency (ABP3)

Parameters:

- __HCLKFREQ__: HCLK frequency
- __APB3PRESCALER__: This parameter can be one of the following values:
  – LL_RCC_APB3_DIV_1
  – LL_RCC_APB3_DIV_2
  – LL_RCC_APB3_DIV_4
  – LL_RCC_APB3_DIV_8
  – LL_RCC_APB3_DIV_16

Return value:

- PCLK3: clock frequency (in Hz)

__LL_RCC_CALC_MSIS_FREQ

Description:

- Helper macro to calculate the MSIS frequency (in Hz)

Parameters:

- __MSISSEL__: This parameter can be one of the following values:
  - LL_RCC_MSIRANGESEL_STANDBY
  - LL_RCC_MSIRANGESEL_RUN
- __MSISRANGE__: This parameter can be one of the following values:
  - LL_RCC_MSISRANGE_0
  - LL_RCC_MSISRANGE_1
  - LL_RCC_MSISRANGE_2
  - LL_RCC_MSISRANGE_3
  - LL_RCC_MSISRANGE_4
  - LL_RCC_MSISRANGE_5
  - LL_RCC_MSISRANGE_6
  - LL_RCC_MSISRANGE_7
  - LL_RCC_MSISRANGE_8
  - LL_RCC_MSISRANGE_9
  - LL_RCC_MSISRANGE_10
  - LL_RCC_MSISRANGE_11
  - LL_RCC_MSISRANGE_12
  - LL_RCC_MSISRANGE_13
  - LL_RCC_MSISRANGE_14
  - LL_RCC_MSISRANGE_15

Return value:

- MSI: clock frequency (in Hz)

Notes:

- __MSISSEL__ can be retrieved thanks to function LL_RCC_MSI_IsEnabledRangeSelect() if __MSISSEL__ is equal to LL_RCC_MSIRANGESEL_STANDBY, __MSISRANGE__can be retrieved by LL_RCC_MSI_GetRangeAfterStandby() else by LL_RCC_MSI_GetRange() ex: __LL_RCC_CALC_MSIS_FREQ(LL_RCC_MSI_IsEnabledRangeSelect(), (LL_RCC_MSI_IsEnabledRangeSelect()? LL_RCC_MSI_GetRange(): LL_RCC_MSI_GetRangeAfterStandby()))

__LL_RCC_CALC_MSIK_FREQ

Description:

- Helper macro to calculate the MSIK frequency (in Hz)

Parameters:

- __MSIKSEL__: This parameter can be one of the following values:
  - LL_RCC_MSIRANGESEL_STANDBY
  - LL_RCC_MSIRANGESEL_RUN
- __MSIKRANGE__: This parameter can be one of the following values:
  - LL_RCC_MSIKRANGE_0
  - LL_RCC_MSIKRANGE_1
  - LL_RCC_MSIKRANGE_2
  - LL_RCC_MSIKRANGE_3
  - LL_RCC_MSIKRANGE_4
  - LL_RCC_MSIKRANGE_5
  - LL_RCC_MSIKRANGE_6
  - LL_RCC_MSIKRANGE_7
  - LL_RCC_MSIKRANGE_8
  - LL_RCC_MSIKRANGE_9
  - LL_RCC_MSIKRANGE_10
  - LL_RCC_MSIKRANGE_11
  - LL_RCC_MSIKRANGE_12
  - LL_RCC_MSIKRANGE_13
  - LL_RCC_MSIKRANGE_14
  - LL_RCC_MSIKRANGE_15

Return value:

- MSIK: clock frequency (in Hz)

Notes:

- __MSIKSEL__ can be retrieved thanks to function LL_RCC_MSIK_IsEnabledRangeSelect() if __MSIKSEL__ is equal to LL_RCC_MSIKRANGESEL_STANDBY, __MSIKRANGE__can be retrieved by LL_RCC_MSIK_GetRangeAfterStandby() else by LL_RCC_MSIK_GetRange() ex: __LL_RCC_CALC_MSIK_FREQ(LL_RCC_MSIK_IsEnabledRangeSelect(), (LL_RCC_MSIK_IsEnabledRangeSelect()? LL_RCC_MSIK_GetRange(): LL_RCC_MSIK_GetRangeAfterStandby()))

Common Write and read registers Macros

LL_RCC_WriteReg

Description:

- Write a value in RCC register.

Parameters:

- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RCC_ReadReg

Description:

- Read a value in RCC register.

Parameters:

- __REG__: Register to be read

Return value:

- Register: value

115 LL RNG Generic Driver

115.1 RNG Firmware driver registers structures

115.1.1 LL_RNG_InitTypeDef

LL_RNG_InitTypeDef is defined in the stm32u5xx_ll_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t LL_RNG_InitTypeDef::ClockErrorDetection***
  Clock error detection. This parameter can be one value of ***RNG_LL_CED***. This parameter can be modified using unitary functions **LL_RNG_EnableClkErrorDetect()**.

115.2 RNG Firmware driver API description

The following section lists the various functions of the RNG library.

115.2.1 Detailed description of functions

LL_RNG_Enable

Function name

__STATIC_INLINE void LL_RNG_Enable (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Enable

LL_RNG_Disable

Function name

__STATIC_INLINE void LL_RNG_Disable (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generation.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_Disable

LL_RNG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabled (const RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR RNGEN LL_RNG_IsEnabled

LL_RNG_EnableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_EnableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Enable Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_EnableClkErrorDetect

LL_RNG_DisableClkErrorDetect

Function name

__STATIC_INLINE void LL_RNG_DisableClkErrorDetect (RNG_TypeDef * RNGx)

Function description

Disable RNG Clock Error Detection.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CED LL_RNG_DisableClkErrorDetect

LL_RNG_IsEnabledClkErrorDetect

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledClkErrorDetect (const RNG_TypeDef * RNGx)

Function description

Check if RNG Clock Error Detection is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CED LL_RNG_IsEnabledClkErrorDetect

LL_RNG_EnableCondReset

Function name

__STATIC_INLINE void LL_RNG_EnableCondReset (RNG_TypeDef * RNGx)

Function description

Set RNG Conditioning Soft Reset bit.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_EnableCondReset

LL_RNG_DisableCondReset

Function name

__STATIC_INLINE void LL_RNG_DisableCondReset (RNG_TypeDef * RNGx)

Function description

Reset RNG Conditioning Soft Reset bit.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_DisableCondReset

LL_RNG_IsEnabledCondReset

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledCondReset (const RNG_TypeDef * RNGx)

Function description

Check if RNG Conditioning Soft Reset bit is set.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CONDRST LL_RNG_IsEnabledCondReset

LL_RNG_ConfigLock

Function name

__STATIC_INLINE void LL_RNG_ConfigLock (RNG_TypeDef * RNGx)

Function description

Enable RNG Config Lock.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CONFIGLOCK LL_RNG_ConfigLock

LL_RNG_IsConfigLocked

Function name

__STATIC_INLINE uint32_t LL_RNG_IsConfigLocked (const RNG_TypeDef * RNGx)

Function description

Check if RNG Config Lock is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR CONFIGLOCK LL_RNG_IsConfigLocked

LL_RNG_EnableNistCompliance

Function name

__STATIC_INLINE void LL_RNG_EnableNistCompliance (RNG_TypeDef * RNGx)

Function description

Enable NIST Compliance.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_EnableNistCompliance

LL_RNG_DisableNistCompliance

Function name

__STATIC_INLINE void LL_RNG_DisableNistCompliance (RNG_TypeDef * RNGx)

Function description

Disable NIST Compliance.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_DisableNistCompliance

LL_RNG_IsEnabledNistCompliance

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledNistCompliance (const RNG_TypeDef * RNGx)

Function description

Check if NIST Compliance is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR NISTC LL_RNG_IsEnabledNistCompliance

LL_RNG_SetConfig1

Function name

__STATIC_INLINE void LL_RNG_SetConfig1 (RNG_TypeDef * RNGx, uint32_t Config1)

Function description

Set RNG Config1 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config1:** Value between 0 and 0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG1 LL_RNG_SetConfig1

LL_RNG_GetConfig1

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig1 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config1 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 6 bits : Value between 0 and 0x3F

Reference Manual to LL API cross reference:

- CR RNG_CONFIG1 LL_RNG_GetConfig1

LL_RNG_SetConfig2

Function name

__STATIC_INLINE void LL_RNG_SetConfig2 (RNG_TypeDef * RNGx, uint32_t Config2)

Function description

Set RNG Config2 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config2:** Value between 0 and 0x7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG2 LL_RNG_SetConfig2

LL_RNG_GetConfig2

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig2 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config2 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 3 bits : Value between 0 and 0x7

Reference Manual to LL API cross reference:

- CR RNG_CONFIG2 LL_RNG_GetConfig2

LL_RNG_SetConfig3

Function name

__STATIC_INLINE void LL_RNG_SetConfig3 (RNG_TypeDef * RNGx, uint32_t Config3)

Function description

Set RNG Config3 Configuration field value.

Parameters

- **RNGx:** RNG Instance
- **Config3:** Value between 0 and 0xF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RNG_CONFIG3 LL_RNG_SetConfig3

LL_RNG_GetConfig3

Function name

__STATIC_INLINE uint32_t LL_RNG_GetConfig3 (const RNG_TypeDef * RNGx)

Function description

Get RNG Config3 Configuration field value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** Value expressed on 4 bits : Value between 0 and 0xF

Reference Manual to LL API cross reference:

- CR RNG_CONFIG3 LL_RNG_GetConfig3

LL_RNG_SetClockDivider

Function name

__STATIC_INLINE void LL_RNG_SetClockDivider (RNG_TypeDef * RNGx, uint32_t Divider)

Function description

Set RNG Clock divider factor.

Parameters

- **RNGx:** RNG Instance
- **Divider:** can be one of the following values:
  - LL_RNG_CLKDIV_BY_1
  - LL_RNG_CLKDIV_BY_2
  - LL_RNG_CLKDIV_BY_4
  - LL_RNG_CLKDIV_BY_8
  - LL_RNG_CLKDIV_BY_16
  - LL_RNG_CLKDIV_BY_32
  - LL_RNG_CLKDIV_BY_64
  - LL_RNG_CLKDIV_BY_128
  - LL_RNG_CLKDIV_BY_256
  - LL_RNG_CLKDIV_BY_512
  - LL_RNG_CLKDIV_BY_1024
  - LL_RNG_CLKDIV_BY_2048
  - LL_RNG_CLKDIV_BY_4096
  - LL_RNG_CLKDIV_BY_8192
  - LL_RNG_CLKDIV_BY_16384
  - LL_RNG_CLKDIV_BY_32768

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CLKDIV LL_RNG_SetClockDivider

LL_RNG_GetClockDivider

Function name

__STATIC_INLINE uint32_t LL_RNG_GetClockDivider (const RNG_TypeDef * RNGx)

Function description

Get RNG Clock divider factor.

Parameters

- **RNGx:** RNG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RNG_CLKDIV_BY_1
  - LL_RNG_CLKDIV_BY_2
  - LL_RNG_CLKDIV_BY_4
  - LL_RNG_CLKDIV_BY_8
  - LL_RNG_CLKDIV_BY_16
  - LL_RNG_CLKDIV_BY_32
  - LL_RNG_CLKDIV_BY_64
  - LL_RNG_CLKDIV_BY_128
  - LL_RNG_CLKDIV_BY_256
  - LL_RNG_CLKDIV_BY_512
  - LL_RNG_CLKDIV_BY_1024
  - LL_RNG_CLKDIV_BY_2048
  - LL_RNG_CLKDIV_BY_4096
  - LL_RNG_CLKDIV_BY_8192
  - LL_RNG_CLKDIV_BY_16384
  - LL_RNG_CLKDIV_BY_32768

Reference Manual to LL API cross reference:

- CR CLKDIV LL_RNG_GetClockDivider

LL_RNG_IsActiveFlag_DRDY

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_DRDY (const RNG_TypeDef * RNGx)

Function description

Indicate if the RNG Data ready Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR DRDY LL_RNG_IsActiveFlag_DRDY

LL_RNG_IsActiveFlag_CECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CECS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CECS LL_RNG_IsActiveFlag_CECS

LL_RNG_IsActiveFlag_SECS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SECS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Current Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SECS LL_RNG_IsActiveFlag_SECS

LL_RNG_IsActiveFlag_CEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_CEIS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Clock Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_IsActiveFlag_CEIS

LL_RNG_IsActiveFlag_SEIS

Function name

__STATIC_INLINE uint32_t LL_RNG_IsActiveFlag_SEIS (const RNG_TypeDef * RNGx)

Function description

Indicate if the Seed Error Interrupt Status Flag is set or not.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_IsActiveFlag_SEIS

LL_RNG_ClearFlag_CEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_CEIS (RNG_TypeDef * RNGx)

Function description

Clear Clock Error interrupt Status (CEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CEIS LL_RNG_ClearFlag_CEIS

LL_RNG_ClearFlag_SEIS

Function name

__STATIC_INLINE void LL_RNG_ClearFlag_SEIS (RNG_TypeDef * RNGx)

Function description

Clear Seed Error interrupt Status (SEIS) Flag.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SEIS LL_RNG_ClearFlag_SEIS

LL_RNG_EnableIT

Function name

__STATIC_INLINE void LL_RNG_EnableIT (RNG_TypeDef * RNGx)

Function description

Enable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_EnableIT

LL_RNG_DisableIT

Function name

__STATIC_INLINE void LL_RNG_DisableIT (RNG_TypeDef * RNGx)

Function description

Disable Random Number Generator Interrupt (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR IE LL_RNG_DisableIT

LL_RNG_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledIT (const RNG_TypeDef * RNGx)

Function description

Check if Random Number Generator Interrupt is enabled (applies for either Seed error, Clock Error or Data ready interrupts)

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR IE LL_RNG_IsEnabledIT

LL_RNG_ReadRandData32

Function name

__STATIC_INLINE uint32_t LL_RNG_ReadRandData32 (const RNG_TypeDef * RNGx)

Function description

Return32-bit Random Number value.

Parameters

- **RNGx:** RNG Instance

Return values

- **Generated:** 32-bit random value

Reference Manual to LL API cross reference:

- DR RNDATA LL_RNG_ReadRandData32

LL_RNG_EnableArdis

Function name

__STATIC_INLINE void LL_RNG_EnableArdis (RNG_TypeDef * RNGx)

Function description

Enable Auto reset.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_EnableArdis

LL_RNG_DisableArdis

Function name

__STATIC_INLINE void LL_RNG_DisableArdis (RNG_TypeDef * RNGx)

Function description

Disable Auto reset.

Parameters

- **RNGx:** RNG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_DisableArdis

LL_RNG_IsEnabledArdis

Function name

__STATIC_INLINE uint32_t LL_RNG_IsEnabledArdis (const RNG_TypeDef * RNGx)

Function description

Check if RNG Auto reset is enabled.

Parameters

- **RNGx:** RNG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR ARDIS LL_RNG_IsEnabledArdis

LL_RNG_SetHealthConfig

Function name

__STATIC_INLINE void LL_RNG_SetHealthConfig (RNG_TypeDef * RNGx, uint32_t HTCFG)

Function description

Set RNG Health Test Control.

Parameters

- **RNGx:** RNG Instance
- **HTCFG:** can be values of 32 bits

Return values

- **None:**

Reference Manual to LL API cross reference:

- HTCR HTCFG LL_RNG_SetHealthConfig

LL_RNG_GetHealthConfig

Function name

__STATIC_INLINE uint32_t LL_RNG_GetHealthConfig (const RNG_TypeDef * RNGx)

Function description

Get RNG Health Test Control.

Parameters

- **RNGx:** RNG Instance

Return values

- **Return:** 32-bit RNG Health Test configuration

Reference Manual to LL API cross reference:

- HTCR HTCFG LL_RNG_GetHealthConfig

LL_RNG_Init

Function name

ErrorStatus LL_RNG_Init (RNG_TypeDef * RNGx, LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Initialize RNG registers according to the specified parameters in RNG_InitStruct.

Parameters

- **RNGx:** RNG Instance
- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure that contains the configuration information for the specified RNG peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RNG registers are initialized according to RNG_InitStruct content
  - ERROR: not applicable

LL_RNG_StructInit

Function name

void LL_RNG_StructInit (LL_RNG_InitTypeDef * RNG_InitStruct)

Function description

Set each LL_RNG_InitTypeDef field to default value.

Parameters

- **RNG_InitStruct:** pointer to a LL_RNG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_RNG_DeInit

Function name

ErrorStatus LL_RNG_DeInit (const RNG_TypeDef * RNGx)

Function description

De-initialize RNG registers (Registers restored to their default values).

Parameters

- **RNGx:** RNG Instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: RNG registers are de-initialized
  – ERROR: not applicable

115.3 RNG Firmware driver defines

The following section lists the various define and macros of the module.

115.3.1 RNG

RNG

Auto reset disable

LL_RNG_ARDIS_ENABLE

ARDIS enabled automatic reset to clear SECS bit

LL_RNG_ARDIS_DISABLE

ARDIS disabled no automatic reset to clear SECS bit

Clock Error Detection

LL_RNG_CED_ENABLE

Clock error detection enabled

LL_RNG_CED_DISABLE

Clock error detection disabled

Value used to configure an internal

LL_RNG_CLKDIV_BY_1

No clock division

LL_RNG_CLKDIV_BY_2

2 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_4

4 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_8

8 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_16

16 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_32

32 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_64

64 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_128

128 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_256

256 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_512

512 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_1024

1024 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_2048

2048 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_4096

4096 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_8192

8192 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_16384

16384 RNG clock cycles per internal RNG clock

LL_RNG_CLKDIV_BY_32768

32768 RNG clock cycles per internal RNG clock

Get Flags Defines

LL_RNG_SR_DRDY

Register contains valid random data

LL_RNG_SR_CECS

Clock error current status

LL_RNG_SR_SECS

Seed error current status

LL_RNG_SR_CEIS

Clock error interrupt status

LL_RNG_SR_SEIS

Seed error interrupt status

IT Defines

LL_RNG_CR_IE

RNG Interrupt enable

Common Write and read registers Macros

LL_RNG_WriteReg

Description:

- Write a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RNG_ReadReg

Description:

- Read a value in RNG register.

Parameters:

- __INSTANCE__: RNG Instance
- __REG__: Register to be read

Return value:

- Register: value

NIST Compliance configuration

LL_RNG_NIST_COMPLIANT

Default NIST compliant configuration

LL_RNG_CUSTOM_NIST

Custom NIST configuration

116 LL RTC Generic Driver

116.1 RTC Firmware driver registers structures

116.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32u5xx_ll_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrescaler***
- ***uint32_t SynchPrescaler***

Field Documentation

- ***uint32_t LL_RTC_InitTypeDef::HourFormat***
  Specifies the RTC Hours Format. This parameter can be a value of ***RTC_LL_EC_HOURFORMAT***This feature can be modified afterwards using unitary function **LL_RTC_SetHourFormat()**.
- ***uint32_t LL_RTC_InitTypeDef::AsynchPrescaler***
  Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FThis feature can be modified afterwards using unitary function **LL_RTC_SetAsynchPrescaler()**.
- ***uint32_t LL_RTC_InitTypeDef::SynchPrescaler***
  Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFFThis feature can be modified afterwards using unitary function **LL_RTC_SetSynchPrescaler()**.

116.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32u5xx_ll_rtc.h

Data Fields

- ***uint32_t TimeFormat***
- ***uint8_t Hours***
- ***uint8_t Minutes***
- ***uint8_t Seconds***

Field Documentation

- ***uint32_t LL_RTC_TimeTypeDef::TimeFormat***
  Specifies the RTC AM/PM Time. This parameter can be a value of ***RTC_LL_EC_TIME_FORMAT*** This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetFormat()**.
- ***uint8_t LL_RTC_TimeTypeDef::Hours***
  Specifies the RTC Time Hours. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the **LL_RTC_TIME_FORMAT_PM** is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the **LL_RTC_TIME_FORMAT_AM_OR_24** is selected. This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetHour()**.
- ***uint8_t LL_RTC_TimeTypeDef::Minutes***
  Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59 This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetMinute()**.
- ***uint8_t LL_RTC_TimeTypeDef::Seconds***
  Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59 This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

116.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32u5xx_ll_rtc.h

Data Fields

- ***uint8_t WeekDay***
- ***uint8_t Month***
- ***uint8_t Day***

- ***uint8_t Year***

Field Documentation

- ***uint8_t LL_RTC_DateTypeDef::WeekDay***
  Specifies the RTC Date WeekDay. This parameter can be a value of ***RTC_LL_EC_WEEKDAY*** This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetWeekDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Month***
  Specifies the RTC Date Month. This parameter can be a value of ***RTC_LL_EC_MONTH*** This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetMonth()**.
- ***uint8_t LL_RTC_DateTypeDef::Day***
  Specifies the RTC Date Day. This parameter must be a number between Min_Data = 1 and Max_Data = 31 This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetDay()**.
- ***uint8_t LL_RTC_DateTypeDef::Year***
  Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99 This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

116.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32u5xx_ll_rtc.h

Data Fields

- ***LL_RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***

Field Documentation

- ***LL_RTC_TimeTypeDef LL_RTC_AlarmTypeDef::AlarmTime***
  Specifies the RTC Alarm Time members.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmMask***
  Specifies the RTC Alarm Masks. This parameter can be a value of ***RTC_LL_EC_ALMA_MASK*** for ALARM A or ***RTC_LL_EC_ALMB_MASK*** for ALARM B. This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A or **LL_RTC_ALMB_SetMask()** for ALARM B
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A or ***RTC_LL_EC_ALMB_WEEKDAY_SELECTION*** for ALARM B This feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A or **LL_RTC_ALMB_EnableWeekday()** or **LL_RTC_ALMB_DisableWeekday()** for ALARM B
- ***uint8_t LL_RTC_AlarmTypeDef::AlarmDateWeekDay***
  Specifies the RTC Alarm Day/WeekDay. If AlarmDateWeekDaySel set to day, this parameter must be a number between Min_Data = 1 and Max_Data = 31. This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetDay()** for ALARM A or **LL_RTC_ALMB_SetDay()** for ALARM B. If AlarmDateWeekDaySel set to Weekday, this parameter can be a value of ***RTC_LL_EC_WEEKDAY***. This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetWeekDay()** for ALARM A or **LL_RTC_ALMB_SetWeekDay()** for ALARM B.

116.2 RTC Firmware driver API description

The following section lists the various functions of the RTC library.

116.2.1 Detailed description of functions

LL_RTC_SetHourFormat

Function name

__STATIC_INLINE void LL_RTC_SetHourFormat (RTC_TypeDef * RTCx, uint32_t HourFormat)

Function description

Set Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance
- **HourFormat:** This parameter can be one of the following values:
  – LL_RTC_HOURFORMAT_24HOUR
  – LL_RTC_HOURFORMAT_AMPM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (const RTC_TypeDef * RTCx)

Function description

Get Hours format (24 hour/day or AM/PM hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_HOURFORMAT_24HOUR
  – LL_RTC_HOURFORMAT_AMPM

Reference Manual to LL API cross reference:

- RTC_CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)

Function description

Select the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance
- **AlarmOutput:** This parameter can be one of the following values:
  – LL_RTC_ALARMOUT_DISABLE
  – LL_RTC_ALARMOUT_ALMA
  – LL_RTC_ALARMOUT_ALMB
  – LL_RTC_ALARMOUT_WAKEUP

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (const RTC_TypeDef * RTCx)

Function description

Get the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Reference Manual to LL API cross reference:

- RTC_CR OSEL LL_RTC_GetAlarmOutEvent

LL_RTC_SetAlarmOutputType

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutputType (RTC_TypeDef * RTCx, uint32_t Output)

Function description

Set RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance
- **Output:** This parameter can be one of the following values:
  - LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  - LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_SetAlarmOutputType

LL_RTC_GetAlarmOutputType

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutputType (const RTC_TypeDef * RTCx)

Function description

Get RTC_ALARM output type (ALARM in push-pull or open-drain output)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN
  – LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_TYPE LL_RTC_SetAlarmOutputType

LL_RTC_EnableInitMode

Function name

__STATIC_INLINE void LL_RTC_EnableInitMode (RTC_TypeDef * RTCx)

Function description

Enable initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Initialization mode is used to program time and date register (RTC_TR and RTC_DR) and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_EnableInitMode

LL_RTC_DisableInitMode

Function name

__STATIC_INLINE void LL_RTC_DisableInitMode (RTC_TypeDef * RTCx)

Function description

Disable initialization mode (Free running mode)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR INIT LL_RTC_DisableInitMode

LL_RTC_SetBinaryMode

Function name

__STATIC_INLINE void LL_RTC_SetBinaryMode (RTC_TypeDef * RTCx, uint32_t BinaryMode)

Function description

Set Binary mode (Sub Second Register)

Parameters

- **RTCx:** RTC Instance
- **BinaryMode:** can be one of the following values:
  – LL_RTC_BINARY_NONE
  – LL_RTC_BINARY_ONLY
  – LL_RTC_BINARY_MIX

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function).

Reference Manual to LL API cross reference:

- RTC_ICSR BIN LL_RTC_SetBinaryMode

LL_RTC_GetBinaryMode

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBinaryMode (const RTC_TypeDef * RTCx)

Function description

Get Binary mode (Sub Second Register)

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  – LL_RTC_BINARY_NONE
  – LL_RTC_BINARY_ONLY
  – LL_RTC_BINARY_MIX
- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR BIN LL_RTC_GetBinaryMode

LL_RTC_SetBinMixBCDU

Function name

__STATIC_INLINE void LL_RTC_SetBinMixBCDU (RTC_TypeDef * RTCx, uint32_t BinMixBcdU)

Function description

Set Binary Mix mode BCDU.

Parameters

- **RTCx:** RTC Instance
- **BinMixBcdU:** can be one of the following values:
  – LL_RTC_BINARY_MIX_BCDU_0
  – LL_RTC_BINARY_MIX_BCDU_1
  – LL_RTC_BINARY_MIX_BCDU_2
  – LL_RTC_BINARY_MIX_BCDU_3
  – LL_RTC_BINARY_MIX_BCDU_4
  – LL_RTC_BINARY_MIX_BCDU_5
  – LL_RTC_BINARY_MIX_BCDU_6
  – LL_RTC_BINARY_MIX_BCDU_7

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function).

Reference Manual to LL API cross reference:

- RTC_ICSR BCDU LL_RTC_SetBinMixBCDU

LL_RTC_GetBinMixBCDU

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBinMixBCDU (const RTC_TypeDef * RTCx)

Function description

Get Binary Mix mode BCDU.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  – LL_RTC_BINARY_MIX_BCDU_0
  – LL_RTC_BINARY_MIX_BCDU_1
  – LL_RTC_BINARY_MIX_BCDU_2
  – LL_RTC_BINARY_MIX_BCDU_3
  – LL_RTC_BINARY_MIX_BCDU_4
  – LL_RTC_BINARY_MIX_BCDU_5
  – LL_RTC_BINARY_MIX_BCDU_6
  – LL_RTC_BINARY_MIX_BCDU_7
- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR BCDU LL_RTC_GetBinMixBCDU

LL_RTC_SetOutputPolarity

Function name

__STATIC_INLINE void LL_RTC_SetOutputPolarity (RTC_TypeDef * RTCx, uint32_t Polarity)

Function description

Set Output polarity (pin is low when ALRAF/ALRBF/WUTF is asserted)

Parameters

- **RTCx:** RTC Instance
- **Polarity:** This parameter can be one of the following values:
  – LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  – LL_RTC_OUTPUTPOLARITY_PIN_LOW

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (const RTC_TypeDef * RTCx)

Function description

Get Output polarity.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  – LL_RTC_OUTPUTPOLARITY_PIN_LOW

Reference Manual to LL API cross reference:

- RTC_CR POL LL_RTC_GetOutputPolarity

LL_RTC_EnableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_EnableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Enable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_EnableShadowRegBypass

LL_RTC_DisableShadowRegBypass

Function name

__STATIC_INLINE void LL_RTC_DisableShadowRegBypass (RTC_TypeDef * RTCx)

Function description

Disable Bypass the shadow registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Shadow registers bypass is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

LL_RTC_EnableRefClock

Function name

__STATIC_INLINE void LL_RTC_EnableRefClock (RTC_TypeDef * RTCx)

Function description

Enable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_EnableRefClock

LL_RTC_DisableRefClock

Function name

__STATIC_INLINE void LL_RTC_DisableRefClock (RTC_TypeDef * RTCx)

Function description

Disable RTC_REFIN reference clock detection (50 or 60 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_CR REFCKON LL_RTC_DisableRefClock

LL_RTC_SetAsynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetAsynchPrescaler (RTC_TypeDef * RTCx, uint32_t AsynchPrescaler)

Function description

Set Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **AsynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7F

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_SetAsynchPrescaler

LL_RTC_SetSynchPrescaler

Function name

__STATIC_INLINE void LL_RTC_SetSynchPrescaler (RTC_TypeDef * RTCx, uint32_t SynchPrescaler)

Function description

Set Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance
- **SynchPrescaler:** Value between Min_Data = 0 and Max_Data = 0x7FFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (const RTC_TypeDef * RTCx)

Function description

Get Asynchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7F

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (const RTC_TypeDef * RTCx)

Function description

Get Synchronous prescaler factor.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data = 0 and Max_Data = 0x7FFF

Reference Manual to LL API cross reference:

- RTC_PRER PREDIV_S LL_RTC_GetSynchPrescaler

LL_RTC_EnableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_EnableWriteProtection (RTC_TypeDef * RTCx)

Function description

Enable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_EnableWriteProtection

LL_RTC_DisableWriteProtection

Function name

__STATIC_INLINE void LL_RTC_DisableWriteProtection (RTC_TypeDef * RTCx)

Function description

Disable the write protection for RTC registers.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_WPR KEY LL_RTC_DisableWriteProtection

LL_RTC_EnableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_EnableTamperOutput (RTC_TypeDef * RTCx)

Function description

Enable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- When the tamper output is enabled, all external and internal tamper flags are ORed and routed to the TAMPALRM output.

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_EnableTamperOutput

LL_RTC_DisableTamperOutput

Function name

__STATIC_INLINE void LL_RTC_DisableTamperOutput (RTC_TypeDef * RTCx)

Function description

Disable tamper output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_DisableTamperOutput

LL_RTC_IsTamperOutputEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsTamperOutputEnabled (const RTC_TypeDef * RTCx)

Function description

Check if tamper output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPOE LL_RTC_IsTamperOutputEnabled

LL_RTC_EnableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_EnableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Enable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_DisableAlarmPullUp

Function name

__STATIC_INLINE void LL_RTC_DisableAlarmPullUp (RTC_TypeDef * RTCx)

Function description

Disable internal pull-up in output mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_EnableAlarmPullUp

LL_RTC_IsAlarmPullUpEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsAlarmPullUpEnabled (const RTC_TypeDef * RTCx)

Function description

Check if internal pull-up in output mode is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TAMPALRM_PU LL_RTC_IsAlarmPullUpEnabled

LL_RTC_EnableOutput2

Function name

__STATIC_INLINE void LL_RTC_EnableOutput2 (RTC_TypeDef * RTCx)

Function description

Enable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- RTC_OUT2 mapping depends on both OSEL (LL_RTC_SetAlarmOutEvent) and COE (LL_RTC_CAL_SetOutputFreq) settings.
- RTC_OUT2 is not available ins VBAT mode.

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_EnableOutput2

LL_RTC_DisableOutput2

Function name

__STATIC_INLINE void LL_RTC_DisableOutput2 (RTC_TypeDef * RTCx)

Function description

Disable RTC_OUT2 output.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_DisableOutput2

LL_RTC_IsOutput2Enabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsOutput2Enabled (const RTC_TypeDef * RTCx)

Function description

Check if RTC_OUT2 output is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR OUT2EN LL_RTC_IsOutput2Enabled

LL_RTC_TIME_SetFormat

Function name

__STATIC_INLINE void LL_RTC_TIME_SetFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  – LL_RTC_TIME_FORMAT_AM_OR_24
  – LL_RTC_TIME_FORMAT_PM

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (const RTC_TypeDef * RTCx)

Function description

Get time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_TIME_FORMAT_AM_OR_24
  – LL_RTC_TIME_FORMAT_PM

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_GetFormat

LL_RTC_TIME_SetHour

Function name

__STATIC_INLINE void LL_RTC_TIME_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert hour from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_SetHour
- RTC_TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (const RTC_TypeDef * RTCx)

Function description

Get Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert hour from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_GetHour
- RTC_TR HU LL_RTC_TIME_GetHour

LL_RTC_TIME_SetMinute

Function name

__STATIC_INLINE void LL_RTC_TIME_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_SetMinute
- RTC_TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert minute from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR MNT LL_RTC_TIME_GetMinute
- RTC_TR MNU LL_RTC_TIME_GetMinute

LL_RTC_TIME_SetSecond

Function name

__STATIC_INLINE void LL_RTC_TIME_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_SetSecond
- RTC_TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TR ST LL_RTC_TIME_GetSecond
- RTC_TR SU LL_RTC_TIME_GetSecond

LL_RTC_TIME_Config

Function name

__STATIC_INLINE void LL_RTC_TIME_Config (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_TIME_FORMAT_AM_OR_24
  - LL_RTC_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- It can be written in initialization mode only (LL_RTC_EnableInitMode function)
- TimeFormat and Hours should follow the same format

Reference Manual to LL API cross reference:

- RTC_TR PM LL_RTC_TIME_Config
- RTC_TR HT LL_RTC_TIME_Config
- RTC_TR HU LL_RTC_TIME_Config
- RTC_TR MNT LL_RTC_TIME_Config
- RTC_TR MNU LL_RTC_TIME_Config
- RTC_TR ST LL_RTC_TIME_Config
- RTC_TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_Get (const RTC_TypeDef * RTCx)

Function description

Get time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds (Format: 0x00HHMMSS).

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- Read either RTC_SSR or RTC_TR locks the values in the higher-order calendar shadow registers until RTC_DR is read (LL_RTC_ReadReg(RTC, DR)).
- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TR HT LL_RTC_TIME_Get
- RTC_TR HU LL_RTC_TIME_Get
- RTC_TR MNT LL_RTC_TIME_Get
- RTC_TR MNU LL_RTC_TIME_Get
- RTC_TR ST LL_RTC_TIME_Get
- RTC_TR SU LL_RTC_TIME_Get

LL_RTC_TIME_EnableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_EnableDayLightStore (RTC_TypeDef * RTCx)

Function description

Memorize whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_EnableDayLightStore

LL_RTC_TIME_DisableDayLightStore

Function name

__STATIC_INLINE void LL_RTC_TIME_DisableDayLightStore (RTC_TypeDef * RTCx)

Function description

Disable memorization whether the daylight saving time change has been performed.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (const RTC_TypeDef * RTCx)

Function description

Check if RTC Day Light Saving stored operation has been enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR BKP LL_RTC_TIME_IsDayLightStoreEnabled

LL_RTC_TIME_DecHour

Function name

__STATIC_INLINE void LL_RTC_TIME_DecHour (RTC_TypeDef * RTCx)

Function description

Subtract 1 hour (winter time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SUB1H LL_RTC_TIME_DecHour

LL_RTC_TIME_IncHour

Function name

__STATIC_INLINE void LL_RTC_TIME_IncHour (RTC_TypeDef * RTCx)

Function description

Add 1 hour (summer time change)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Sub second value in the synchronous prescaler counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Notes

- You can use both SubSeconds value and SecondFraction (PREDIV_S through LL_RTC_GetSynchPrescaler function) terms returned to convert Calendar SubSeconds value in second fraction ratio with time unit following generic formula: ==> Seconds fraction ratio * time_unit= [(SecondFraction-SubSeconds)/(SecondFraction+1)] * time_unit This conversion can be performed only if no shift operation is pending (ie. SHFP=0) when PREDIV_S >= SS.

Reference Manual to LL API cross reference:

- RTC_SSR SS LL_RTC_TIME_GetSubSecond

LL_RTC_TIME_Synchronize

Function name

__STATIC_INLINE void LL_RTC_TIME_Synchronize (RTC_TypeDef * RTCx, uint32_t ShiftSecond, uint32_t Fraction)

Function description

Synchronize to a remote clock with a high degree of precision.

Parameters

- **RTCx:** RTC Instance
- **ShiftSecond:** This parameter can be one of the following values:
  – LL_RTC_SHIFT_SECOND_DELAY
  – LL_RTC_SHIFT_SECOND_ADVANCE
- **Fraction:** Number of Seconds Fractions (any value from 0 to 0x7FFF)

Return values

- **None:**

Notes

- This operation effectively subtracts from (delays) or advance the clock of a fraction of a second.
- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- When REFCKON is set, firmware must not write to Shift control register.

Reference Manual to LL API cross reference:

- RTC_SHIFTR ADD1S LL_RTC_TIME_Synchronize
- RTC_SHIFTR SUBFS LL_RTC_TIME_Synchronize

LL_RTC_DATE_SetYear

Function name

__STATIC_INLINE void LL_RTC_DATE_SetYear (RTC_TypeDef * RTCx, uint32_t Year)

Function description

Set Year in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Year from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_SetYear
- RTC_DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (const RTC_TypeDef * RTCx)

Function description

Get Year in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x99

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Year from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR YT LL_RTC_DATE_GetYear
- RTC_DR YU LL_RTC_DATE_GetYear

LL_RTC_DATE_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set Week day.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_GetWeekDay

LL_RTC_DATE_SetMonth

Function name

__STATIC_INLINE void LL_RTC_DATE_SetMonth (RTC_TypeDef * RTCx, uint32_t Month)

Function description

Set Month in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Month:** This parameter can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Month from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_SetMonth
- RTC_DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (const RTC_TypeDef * RTCx)

Function description

Get Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR MT LL_RTC_DATE_GetMonth
- RTC_DR MU LL_RTC_DATE_GetMonth

LL_RTC_DATE_SetDay

Function name

__STATIC_INLINE void LL_RTC_DATE_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_SetDay
- RTC_DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (const RTC_TypeDef * RTCx)

Function description

Get Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_DR DT LL_RTC_DATE_GetDay
- RTC_DR DU LL_RTC_DATE_GetDay

LL_RTC_DATE_Config

Function name

__STATIC_INLINE void LL_RTC_DATE_Config (RTC_TypeDef * RTCx, uint32_t WeekDay, uint32_t Day, uint32_t Month, uint32_t Year)

Function description

Set date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31
- **Month:** This parameter can be one of the following values:
  – LL_RTC_MONTH_JANUARY
  – LL_RTC_MONTH_FEBRUARY
  – LL_RTC_MONTH_MARCH
  – LL_RTC_MONTH_APRIL
  – LL_RTC_MONTH_MAY
  – LL_RTC_MONTH_JUNE
  – LL_RTC_MONTH_JULY
  – LL_RTC_MONTH_AUGUST
  – LL_RTC_MONTH_SEPTEMBER
  – LL_RTC_MONTH_OCTOBER
  – LL_RTC_MONTH_NOVEMBER
  – LL_RTC_MONTH_DECEMBER
- **Year:** Value between Min_Data=0x00 and Max_Data=0x99

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Config
- RTC_DR MT LL_RTC_DATE_Config
- RTC_DR MU LL_RTC_DATE_Config
- RTC_DR DT LL_RTC_DATE_Config
- RTC_DR DU LL_RTC_DATE_Config
- RTC_DR YT LL_RTC_DATE_Config
- RTC_DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_Get (const RTC_TypeDef * RTCx)

Function description

Get date (WeekDay, Day, Month and Year) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of WeekDay, Day, Month and Year (Format: 0xWWDDMMYY).

Notes

- if RTC shadow registers are not bypassed (BYPSHAD=0), need to check if RSF flag is set before reading this bit
- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_YEAR, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_DR WDU LL_RTC_DATE_Get
- RTC_DR MT LL_RTC_DATE_Get
- RTC_DR MU LL_RTC_DATE_Get
- RTC_DR DT LL_RTC_DATE_Get
- RTC_DR DU LL_RTC_DATE_Get
- RTC_DR YT LL_RTC_DATE_Get
- RTC_DR YU LL_RTC_DATE_Get

LL_RTC_ALMA_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Enable

LL_RTC_ALMA_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMA_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm A.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAE LL_RTC_ALMA_Disable

LL_RTC_ALMA_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm A masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  – LL_RTC_ALMA_MASK_NONE
  – LL_RTC_ALMA_MASK_DATEWEEKDAY
  – LL_RTC_ALMA_MASK_HOURS
  – LL_RTC_ALMA_MASK_MINUTES
  – LL_RTC_ALMA_MASK_SECONDS
  – LL_RTC_ALMA_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_SetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (const RTC_TypeDef * RTCx)

Function description

Get the Alarm A masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  – LL_RTC_ALMA_MASK_NONE
  – LL_RTC_ALMA_MASK_DATEWEEKDAY
  – LL_RTC_ALMA_MASK_HOURS
  – LL_RTC_ALMA_MASK_MINUTES
  – LL_RTC_ALMA_MASK_SECONDS
  – LL_RTC_ALMA_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMAR MSK4 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK3 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK2 LL_RTC_ALMA_GetMask
- RTC_ALRMAR MSK1 LL_RTC_ALMA_GetMask

LL_RTC_ALMA_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmA Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_EnableWeekday

LL_RTC_ALMA_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMA_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmA Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR WDSEL LL_RTC_ALMA_DisableWeekday

LL_RTC_ALMA_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_SetDay
- RTC_ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR DT LL_RTC_ALMA_GetDay
- RTC_ALRMAR DU LL_RTC_ALMA_GetDay

LL_RTC_ALMA_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMAR DU LL_RTC_ALMA_GetWeekDay

LL_RTC_ALMA_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set Alarm A time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  – LL_RTC_ALMA_TIME_FORMAT_AM
  – LL_RTC_ALMA_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get Alarm A time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_GetTimeFormat

LL_RTC_ALMA_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_SetHour
- RTC_ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetHour
- RTC_ALRMAR HU LL_RTC_ALMA_GetHour

LL_RTC_ALMA_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_SetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR MNT LL_RTC_ALMA_GetMinute
- RTC_ALRMAR MNU LL_RTC_ALMA_GetMinute

LL_RTC_ALMA_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_SetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get ALARM A Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMAR ST LL_RTC_ALMA_GetSecond
- RTC_ALRMAR SU LL_RTC_ALMA_GetSecond

LL_RTC_ALMA_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMA_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm A Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMA_TIME_FORMAT_AM
  - LL_RTC_ALMA_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMAR PM LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR HU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNT LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR MNU LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR ST LL_RTC_ALMA_ConfigTime
- RTC_ALRMAR SU LL_RTC_ALMA_ConfigTime

LL_RTC_ALMA_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMAR HT LL_RTC_ALMA_GetTime
- RTC_ALRMAR HU LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNT LL_RTC_ALMA_GetTime
- RTC_ALRMAR MNU LL_RTC_ALMA_GetTime
- RTC_ALRMAR ST LL_RTC_ALMA_GetTime
- RTC_ALRMAR SU LL_RTC_ALMA_GetTime

LL_RTC_ALMA_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

LL_RTC_ALMA_SetBinAutoClr

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetBinAutoClr (RTC_TypeDef * RTCx, uint32_t BinaryAutoClr)

Function description

Set Alarm A Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance
- **BinaryAutoClr:** This parameter can be one of the following values:
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

Return values

- **None:**

Notes

- This register can be written only when ALRAE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRABINR SSCLR LL_RTC_ALMA_SetBinAutoClr

LL_RTC_ALMA_GetBinAutoClr

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetBinAutoClr (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance

Return values

- **It:** can be one of the following values:
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO
  - LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

Reference Manual to LL API cross reference:

- RTC_ALRABINR SSCLR LL_RTC_ALMA_GetBinAutoClr

LL_RTC_ALMA_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMA_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Alarm A Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_ALMB_Enable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Enable (RTC_TypeDef * RTCx)

Function description

Enable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Enable

LL_RTC_ALMB_Disable

Function name

__STATIC_INLINE void LL_RTC_ALMB_Disable (RTC_TypeDef * RTCx)

Function description

Disable Alarm B.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBE LL_RTC_ALMB_Disable

LL_RTC_ALMB_SetMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Specify the Alarm B masks.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_SetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_SetMask

LL_RTC_ALMB_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMask (const RTC_TypeDef * RTCx)

Function description

Get the Alarm B masks.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_RTC_ALMB_MASK_NONE
  - LL_RTC_ALMB_MASK_DATEWEEKDAY
  - LL_RTC_ALMB_MASK_HOURS
  - LL_RTC_ALMB_MASK_MINUTES
  - LL_RTC_ALMB_MASK_SECONDS
  - LL_RTC_ALMB_MASK_ALL

Reference Manual to LL API cross reference:

- RTC_ALRMBR MSK4 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK3 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK2 LL_RTC_ALMB_GetMask
- RTC_ALRMBR MSK1 LL_RTC_ALMB_GetMask

LL_RTC_ALMB_EnableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_EnableWeekday (RTC_TypeDef * RTCx)

Function description

Enable AlarmB Week day selection (DU[3:0] represents the week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_EnableWeekday

LL_RTC_ALMB_DisableWeekday

Function name

__STATIC_INLINE void LL_RTC_ALMB_DisableWeekday (RTC_TypeDef * RTCx)

Function description

Disable AlarmB Week day selection (DU[3:0] represents the date)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR WDSEL LL_RTC_ALMB_DisableWeekday

LL_RTC_ALMB_SetDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetDay (RTC_TypeDef * RTCx, uint32_t Day)

Function description

Set ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Day:** Value between Min_Data=0x01 and Max_Data=0x31

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Day from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_SetDay
- RTC_ALRMBR DU LL_RTC_ALMB_SetDay

LL_RTC_ALMB_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR DT LL_RTC_ALMB_GetDay
- RTC_ALRMBR DU LL_RTC_ALMB_GetDay

LL_RTC_ALMB_SetWeekDay

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetWeekDay (RTC_TypeDef * RTCx, uint32_t WeekDay)

Function description

Set ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance
- **WeekDay:** This parameter can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_SetWeekDay

LL_RTC_ALMB_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Weekday.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_WEEKDAY_MONDAY
  – LL_RTC_WEEKDAY_TUESDAY
  – LL_RTC_WEEKDAY_WEDNESDAY
  – LL_RTC_WEEKDAY_THURSDAY
  – LL_RTC_WEEKDAY_FRIDAY
  – LL_RTC_WEEKDAY_SATURDAY
  – LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_ALRMBR DU LL_RTC_ALMB_GetWeekDay

LL_RTC_ALMB_SetTimeFormat

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetTimeFormat (RTC_TypeDef * RTCx, uint32_t TimeFormat)

Function description

Set ALARM B time format (AM/24-hour or PM notation)

Parameters

- **RTCx:** RTC Instance
- **TimeFormat:** This parameter can be one of the following values:
  – LL_RTC_ALMB_TIME_FORMAT_AM
  – LL_RTC_ALMB_TIME_FORMAT_PM

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_SetTimeFormat

LL_RTC_ALMB_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get ALARM B time format (AM or PM notation)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_ALMB_TIME_FORMAT_AM
  – LL_RTC_ALMB_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_GetTimeFormat

LL_RTC_ALMB_SetHour

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetHour (RTC_TypeDef * RTCx, uint32_t Hours)

Function description

Set ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Hours from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_SetHour
- RTC_ALRMBR HU LL_RTC_ALMB_SetHour

LL_RTC_ALMB_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetHour (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetHour
- RTC_ALRMBR HU LL_RTC_ALMB_GetHour

LL_RTC_ALMB_SetMinute

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetMinute (RTC_TypeDef * RTCx, uint32_t Minutes)

Function description

Set ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Minutes:** between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Minutes from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_SetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_SetMinute

LL_RTC_ALMB_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR MNT LL_RTC_ALMB_GetMinute
- RTC_ALRMBR MNU LL_RTC_ALMB_GetMinute

LL_RTC_ALMB_SetSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSecond (RTC_TypeDef * RTCx, uint32_t Seconds)

Function description

Set ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Notes

- helper macro __LL_RTC_CONVERT_BIN2BCD is available to convert Seconds from binary to BCD format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_SetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_SetSecond

LL_RTC_ALMB_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get ALARM B Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_ALRMBR ST LL_RTC_ALMB_GetSecond
- RTC_ALRMBR SU LL_RTC_ALMB_GetSecond

LL_RTC_ALMB_ConfigTime

Function name

__STATIC_INLINE void LL_RTC_ALMB_ConfigTime (RTC_TypeDef * RTCx, uint32_t Format12_24, uint32_t Hours, uint32_t Minutes, uint32_t Seconds)

Function description

Set Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance
- **Format12_24:** This parameter can be one of the following values:
  - LL_RTC_ALMB_TIME_FORMAT_AM
  - LL_RTC_ALMB_TIME_FORMAT_PM
- **Hours:** Value between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23
- **Minutes:** Value between Min_Data=0x00 and Max_Data=0x59
- **Seconds:** Value between Min_Data=0x00 and Max_Data=0x59

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBR PM LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR HU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNT LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR MNU LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR ST LL_RTC_ALMB_ConfigTime
- RTC_ALRMBR SU LL_RTC_ALMB_ConfigTime

LL_RTC_ALMB_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_ALRMBR HT LL_RTC_ALMB_GetTime
- RTC_ALRMBR HU LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNT LL_RTC_ALMB_GetTime
- RTC_ALRMBR MNU LL_RTC_ALMB_GetTime
- RTC_ALRMBR ST LL_RTC_ALMB_GetTime
- RTC_ALRMBR SU LL_RTC_ALMB_GetTime

LL_RTC_ALMB_SetSubSecondMask

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecondMask (RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Set Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance
- **Mask:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_SetSubSecondMask

LL_RTC_ALMB_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecondMask (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Mask the most-significant bits starting at this bit.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0xF else Value between Min_Data=0x0 and Max_Data=0x3F

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR MASKSS LL_RTC_ALMB_GetSubSecondMask

LL_RTC_ALMB_SetBinAutoClr

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetBinAutoClr (RTC_TypeDef * RTCx, uint32_t BinaryAutoClr)

Function description

Set Alarm B Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance
- **BinaryAutoClr:** This parameter can be one of the following values:
  – LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO
  – LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

Return values

- **None:**

Notes

- This register can be written only when ALRBE is reset in RTC_CR register, or in initialization mode.

Reference Manual to LL API cross reference:

- RTC_ALRBBINR SSCLR LL_RTC_ALMB_SetBinAutoClr

LL_RTC_ALMB_GetBinAutoClr

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetBinAutoClr (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Binary mode auto clear.

Parameters

- **RTCx:** RTC Instance

Return values

- **It:** can be one of the following values:
  – LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO
  – LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

Reference Manual to LL API cross reference:

- RTC_ALRBBINR SSCLR LL_RTC_ALMB_GetBinAutoClr

LL_RTC_ALMB_SetSubSecond

Function name

__STATIC_INLINE void LL_RTC_ALMB_SetSubSecond (RTC_TypeDef * RTCx, uint32_t Subsecond)

Function description

Set Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance
- **Subsecond:** If binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_SetSubSecond

LL_RTC_ALMB_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMB_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get Alarm B Sub seconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RTC_ALRMBSSR SS LL_RTC_ALMB_GetSubSecond

LL_RTC_TS_EnableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_EnableInternalEvent (RTC_TypeDef * RTCx)

Function description

Enable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_EnableInternalEvent

LL_RTC_TS_DisableInternalEvent

Function name

__STATIC_INLINE void LL_RTC_TS_DisableInternalEvent (RTC_TypeDef * RTCx)

Function description

Disable internal event timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ITSE LL_RTC_TS_DisableInternalEvent

LL_RTC_TS_Enable

Function name

__STATIC_INLINE void LL_RTC_TS_Enable (RTC_TypeDef * RTCx)

Function description

Enable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Enable

LL_RTC_TS_Disable

Function name

__STATIC_INLINE void LL_RTC_TS_Disable (RTC_TypeDef * RTCx)

Function description

Disable Timestamp.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSE LL_RTC_TS_Disable

LL_RTC_TS_SetActiveEdge

Function name

__STATIC_INLINE void LL_RTC_TS_SetActiveEdge (RTC_TypeDef * RTCx, uint32_t Edge)

Function description

Set Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance
- **Edge:** This parameter can be one of the following values:
  – LL_RTC_TIMESTAMP_EDGE_RISING
  – LL_RTC_TIMESTAMP_EDGE_FALLING

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- TSE must be reset when TSEDGE is changed to avoid unwanted TSF setting

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_SetActiveEdge

LL_RTC_TS_GetActiveEdge

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetActiveEdge (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp event active edge.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_TIMESTAMP_EDGE_RISING
  – LL_RTC_TIMESTAMP_EDGE_FALLING

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSEDGE LL_RTC_TS_GetActiveEdge

LL_RTC_TS_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTimeFormat (const RTC_TypeDef * RTCx)

Function description

Get Timestamp AM/PM notation (AM or 24-hour format)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_TS_TIME_FORMAT_AM
  – LL_RTC_TS_TIME_FORMAT_PM

Reference Manual to LL API cross reference:

- RTC_TSTR PM LL_RTC_TS_GetTimeFormat

LL_RTC_TS_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetHour (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Hours in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x12 or between Min_Data=0x00 and Max_Data=0x23

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Hours from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetHour
- RTC_TSTR HU LL_RTC_TS_GetHour

LL_RTC_TS_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMinute (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Minutes in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Minutes from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR MNT LL_RTC_TS_GetMinute
- RTC_TSTR MNU LL_RTC_TS_GetMinute

LL_RTC_TS_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSecond (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Seconds in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x59

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Seconds from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSTR ST LL_RTC_TS_GetSecond
- RTC_TSTR SU LL_RTC_TS_GetSecond

LL_RTC_TS_GetTime

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetTime (const RTC_TypeDef * RTCx)

Function description

Get Timestamp time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSTR HT LL_RTC_TS_GetTime
- RTC_TSTR HU LL_RTC_TS_GetTime
- RTC_TSTR MNT LL_RTC_TS_GetTime
- RTC_TSTR MNU LL_RTC_TS_GetTime
- RTC_TSTR ST LL_RTC_TS_GetTime
- RTC_TSTR SU LL_RTC_TS_GetTime

LL_RTC_TS_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetWeekDay (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Week day.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetWeekDay

LL_RTC_TS_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetMonth (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Month in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Month from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR MT LL_RTC_TS_GetMonth
- RTC_TSDR MU LL_RTC_TS_GetMonth

LL_RTC_TS_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDay (const RTC_TypeDef * RTCx)

Function description

Get Timestamp Day in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x01 and Max_Data=0x31

Notes

- helper macro __LL_RTC_CONVERT_BCD2BIN is available to convert Day from BCD to Binary format

Reference Manual to LL API cross reference:

- RTC_TSDR DT LL_RTC_TS_GetDay
- RTC_TSDR DU LL_RTC_TS_GetDay

LL_RTC_TS_GetDate

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetDate (const RTC_TypeDef * RTCx)

Function description

Get Timestamp date (WeekDay, Day and Month) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of Weekday, Day and Month

Notes

- helper macros __LL_RTC_GET_WEEKDAY, __LL_RTC_GET_MONTH, and __LL_RTC_GET_DAY are available to get independently each parameter.

Reference Manual to LL API cross reference:

- RTC_TSDR WDU LL_RTC_TS_GetDate
- RTC_TSDR MT LL_RTC_TS_GetDate
- RTC_TSDR MU LL_RTC_TS_GetDate
- RTC_TSDR DT LL_RTC_TS_GetDate
- RTC_TSDR DU LL_RTC_TS_GetDate

LL_RTC_TS_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TS_GetSubSecond (const RTC_TypeDef * RTCx)

Function description

Get time-stamp sub second value.

Parameters

- **RTCx:** RTC Instance

Return values

- **If:** binary mode is none, Value between Min_Data=0x0 and Max_Data=0x7FFF else Value between Min_Data=0x0 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- RTC_TSDR SS LL_RTC_TS_GetSubSecond

LL_RTC_TS_EnableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_EnableOnTamper (RTC_TypeDef * RTCx)

Function description

Activate timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_EnableOnTamper

LL_RTC_TS_DisableOnTamper

Function name

__STATIC_INLINE void LL_RTC_TS_DisableOnTamper (RTC_TypeDef * RTCx)

Function description

Disable timestamp on tamper detection event.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_CR TAMPTS LL_RTC_TS_DisableOnTamper

LL_RTC_TAMPER_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Enable (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Enable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Enable
-

LL_RTC_TAMPER_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_Disable (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Clear TAMPx input detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 TAMP1E LL_RTC_TAMPER_Disable
- TAMP_CR1 TAMP2E... LL_RTC_TAMPER_Disable

LL_RTC_TAMPER_EnableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableMask (const RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Enable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - TAMPER MASK

Return values

- **None:**

Notes

- Associated Tamper IT must not enabled when tamper mask is set.

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_EnableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_EnableMask

LL_RTC_TAMPER_DisableMask

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableMask (const RTC_TypeDef * RTCx, uint32_t Mask)

Function description

Disable Tamper mask flag.

Parameters

- **RTCx:** RTC Instance
- **Mask:** This parameter can be a combination of the following values:
  - TAMPER MASK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1MF LL_RTC_TAMPER_DisableMask
- TAMP_CR2 TAMP2MF... LL_RTC_TAMPER_DisableMask

LL_RTC_TAMPER_EnableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableEraseBKP (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1NOERASE LL_RTC_TAMPER_EnableEraseBKP
- TAMP_CR2 TAMP2NOERASE... LL_RTC_TAMPER_EnableEraseBKP

LL_RTC_TAMPER_DisableEraseBKP

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableEraseBKP (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable backup register erase after Tamper event detection.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  – TAMPER NO ERASE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1NOERASE LL_RTC_TAMPER_DisableEraseBKP
- TAMP_CR2 TAMP2NOERASE... LL_RTC_TAMPER_DisableEraseBKP

LL_RTC_TAMPER_DisablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisablePullUp (const RTC_TypeDef * RTCx)

Function description

Disable RTC_TAMPx pull-up disable (Disable precharge of RTC_TAMPx pins)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_DisablePullUp

LL_RTC_TAMPER_EnablePullUp

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnablePullUp (const RTC_TypeDef * RTCx)

Function description

Enable RTC_TAMPx pull-up disable (Precharge RTC_TAMPx pins before sampling)

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPUDIS LL_RTC_TAMPER_EnablePullUp

LL_RTC_TAMPER_SetPrecharge

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetPrecharge (const RTC_TypeDef * RTCx, uint32_t Duration)

Function description

Set RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance
- **Duration:** This parameter can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_SetPrecharge

LL_RTC_TAMPER_GetPrecharge

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetPrecharge (const RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx precharge duration.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_DURATION_1RTCCLK
  - LL_RTC_TAMPER_DURATION_2RTCCLK
  - LL_RTC_TAMPER_DURATION_4RTCCLK
  - LL_RTC_TAMPER_DURATION_8RTCCLK

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPPRCH LL_RTC_TAMPER_GetPrecharge

LL_RTC_TAMPER_SetFilterCount

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetFilterCount (const RTC_TypeDef * RTCx, uint32_t FilterCount)

Function description

Set RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance
- **FilterCount:** This parameter can be one of the following values:
  – LL_RTC_TAMPER_FILTER_DISABLE
  – LL_RTC_TAMPER_FILTER_2SAMPLE
  – LL_RTC_TAMPER_FILTER_4SAMPLE
  – LL_RTC_TAMPER_FILTER_8SAMPLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_SetFilterCount

LL_RTC_TAMPER_GetFilterCount

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetFilterCount (const RTC_TypeDef * RTCx)

Function description

Get RTC_TAMPx filter count.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_TAMPER_FILTER_DISABLE
  – LL_RTC_TAMPER_FILTER_2SAMPLE
  – LL_RTC_TAMPER_FILTER_4SAMPLE
  – LL_RTC_TAMPER_FILTER_8SAMPLE

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFLT LL_RTC_TAMPER_GetFilterCount

LL_RTC_TAMPER_SetSamplingFreq

Function name

__STATIC_INLINE void LL_RTC_TAMPER_SetSamplingFreq (const RTC_TypeDef * RTCx, uint32_t SamplingFreq)

Function description

Set Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance
- **SamplingFreq:** This parameter can be one of the following values:
  – LL_RTC_TAMPER_SAMPLFREQDIV_32768
  – LL_RTC_TAMPER_SAMPLFREQDIV_16384
  – LL_RTC_TAMPER_SAMPLFREQDIV_8192
  – LL_RTC_TAMPER_SAMPLFREQDIV_4096
  – LL_RTC_TAMPER_SAMPLFREQDIV_2048
  – LL_RTC_TAMPER_SAMPLFREQDIV_1024
  – LL_RTC_TAMPER_SAMPLFREQDIV_512
  – LL_RTC_TAMPER_SAMPLFREQDIV_256

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_SetSamplingFreq

LL_RTC_TAMPER_GetSamplingFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_GetSamplingFreq (const RTC_TypeDef * RTCx)

Function description

Get Tamper sampling frequency.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_TAMPER_SAMPLFREQDIV_32768
  - LL_RTC_TAMPER_SAMPLFREQDIV_16384
  - LL_RTC_TAMPER_SAMPLFREQDIV_8192
  - LL_RTC_TAMPER_SAMPLFREQDIV_4096
  - LL_RTC_TAMPER_SAMPLFREQDIV_2048
  - LL_RTC_TAMPER_SAMPLFREQDIV_1024
  - LL_RTC_TAMPER_SAMPLFREQDIV_512
  - LL_RTC_TAMPER_SAMPLFREQDIV_256

Reference Manual to LL API cross reference:

- TAMP_FLTCR TAMPFREQ LL_RTC_TAMPER_GetSamplingFreq

LL_RTC_TAMPER_EnableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_EnableActiveLevel (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Enable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER ACTIVE LEVEL

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_EnableActiveLevel
- TAMP_CR2 TAMP2TRG... LL_RTC_TAMPER_EnableActiveLevel

LL_RTC_TAMPER_DisableActiveLevel

Function name

__STATIC_INLINE void LL_RTC_TAMPER_DisableActiveLevel (const RTC_TypeDef * RTCx, uint32_t Tamper)

Function description

Disable Active level for Tamper input.

Parameters

- **RTCx:** RTC Instance
- **Tamper:** This parameter can be a combination of the following values:
  - TAMPER ACTIVE LEVEL

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR2 TAMP1TRG LL_RTC_TAMPER_DisableActiveLevel
- TAMP_CR2 TAMP2TRG... LL_RTC_TAMPER_DisableActiveLevel

LL_RTC_TAMPER_ITAMP_Enable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Enable (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Enable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Enable
- TAMP_CR1 ITAMP2E.. LL_RTC_TAMPER_ITAMP_Enable
-

LL_RTC_TAMPER_ITAMP_Disable

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ITAMP_Disable (const RTC_TypeDef * RTCx, uint32_t InternalTamper)

Function description

Disable internal tamper detection.

Parameters

- **RTCx:** RTC Instance
- **InternalTamper:** This parameter can be a combination of the following values:
  - INTERNAL TAMPER

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_CR1 ITAMP1E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP2E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP3E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP5E LL_RTC_TAMPER_ITAMP_Disable
- TAMP_CR1 ITAMP8E LL_RTC_TAMPER_ITAMP_Disable

LL_RTC_TAMPER_ATAMP_EnableActiveMode

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableActiveMode (uint32_t Tamper)

Function description

Enable tamper active mode.

Parameters

- **Tamper:** to configure as active. This parameter can be a combination of the following values:
  – ACTIVE TAMPER MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 TAMP1AM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-
- TAMP_ATCR1 TAMP2AM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-
- TAMP_ATCR1 TAMPxAM LL_RTC_TAMPER_ATAMP_EnableActiveMode
-

LL_RTC_TAMPER_ATAMP_DisableActiveMode

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableActiveMode (uint32_t Tamper)

Function description

Disable tamper active mode.

Parameters

- **Tamper:** to configure as active. This parameter can be a combination of the following values:
  – ACTIVE TAMPER MODE

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 TAMP1AM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-
- TAMP_ATCR1 TAMP2AM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-
- TAMP_ATCR1 TAMPxAM LL_RTC_TAMPER_ATAMP_DisableActiveMode
-

LL_RTC_TAMPER_ATAMP_EnableFilter

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableFilter (void)

Function description

Enable active tamper filter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 FLTEN LL_RTC_TAMPER_ATAMP_EnableFilter
-

LL_RTC_TAMPER_ATAMP_DisableFilter

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableFilter (void)

Function description

Disable active tamper filter.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 FLTEN LL_RTC_TAMPER_ATAMP_DisableFilter
-

LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod (uint32_t ActiveOutputChangePeriod)

Function description

Set Active tamper output change period.

Parameters

- **ActiveOutputChangePeriod:** This parameter can be a value from 0 to 7

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATPER LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod
-

LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod (void)

Function description

Get Active tamper output change period.

Return values

- **Output:** change period. This parameter can be a value from 0 to 7.

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATPER LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod
-

LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler (uint32_t ActiveAsynvPrescaler)

Function description

Set Active tamper asynchronous prescaler clock selection.

Parameters

- **ActiveAsynvPrescaler:** Specifies the Active Tamper asynchronous Prescaler clock. This parameter can be a value of the following values:
  – ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATCKSEL LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler (void)

Function description

Get Active tamper asynchronous prescaler clock selection.

Return values

- **One:** of
  – ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATCKSEL LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_EnableOutputSharing

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_EnableOutputSharing (void)

Function description

Enable active tamper output sharing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATOSHARE LL_RTC_TAMPER_ATAMP_EnableOutputSharing
-

LL_RTC_TAMPER_ATAMP_DisableOutputSharing

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_DisableOutputSharing (void)

Function description

Disable active tamper output sharing.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR1 ATOSHARE LL_RTC_TAMPER_ATAMP_DisableOutputSharing
-

LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection (uint32_t OutputSelection)

Function description

Set Active tamper output selection.

Parameters

- **OutputSelection:** Specifies all the output selection of the Active Tamper. This parameter is a combinasation of the following values: One of
  – ACTIVE TAMPER OUTPUT SELECTION

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATCR2 ATCKSEL LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection
-

LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection

Function name

__STATIC_INLINE uint32_t LL_RTC_TAMPER_ATAMP_GetSharedOuputSelection (void)

Function description

Get Active tamper asynchronous prescaler clock selection.

Return values

- **A:** combination of
  – ACTIVE TAMPER OUTPUT SELECTION

Reference Manual to LL API cross reference:

- TAMP_ATCR2 ATCKSEL LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
-

LL_RTC_TAMPER_ATAMP_WriteSeed

Function name

__STATIC_INLINE void LL_RTC_TAMPER_ATAMP_WriteSeed (uint32_t Seed)

Function description

Write active tamper seed.

Parameters

- **Seed:** Pseudo-random generator seed value

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_ATSEEDR SEED LL_RTC_TAMPER_ATAMP_WriteSeed
-

LL_RTC_IsActiveFlag_ATAMP_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ATAMP_INITS (void)

Function description

Get active tamper initialization status flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_ATOR INITS LL_RTC_IsActiveFlag_ATAMP_INITS

LL_RTC_IsActiveFlag_ATAMP_SEEDF

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ATAMP_SEEDF (void)

Function description

Get active tamper seed running status flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_ATOR INITS LL_RTC_IsActiveFlag_ATAMP_INITS

LL_RTC_WAKEUP_Enable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)

Function description

Select Wakeup clock.

Parameters

- **RTCx:** RTC Instance
- **WakeupClock:** This parameter can be one of the following values:
  – LL_RTC_WAKEUPCLOCK_DIV_16
  – LL_RTC_WAKEUPCLOCK_DIV_8
  – LL_RTC_WAKEUPCLOCK_DIV_4
  – LL_RTC_WAKEUPCLOCK_DIV_2
  – LL_RTC_WAKEUPCLOCK_CKSPRE
  – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ICSR WUTWF bit = 1

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (const RTC_TypeDef * RTCx)

Function description

Get Wakeup clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_WAKEUPCLOCK_DIV_16
  – LL_RTC_WAKEUPCLOCK_DIV_8
  – LL_RTC_WAKEUPCLOCK_DIV_4
  – LL_RTC_WAKEUPCLOCK_DIV_2
  – LL_RTC_WAKEUPCLOCK_CKSPRE
  – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Reference Manual to LL API cross reference:

- RTC_CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)

Function description

Set Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance
- **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Bit can be written only when WUTWF is set to 1 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (const RTC_TypeDef * RTCx)

Function description

Get Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BKP_SetRegister

Function name

__STATIC_INLINE void LL_RTC_BKP_SetRegister (const RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified Backup data register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  – LL_RTC_BKP_DR0
  – LL_RTC_BKP_DR1
  – LL_RTC_BKP_DR2
  – LL_RTC_BKP_DR3
  – LL_RTC_BKP_DR4
  – LL_RTC_BKP_DR5
  – LL_RTC_BKP_DR6
  – LL_RTC_BKP_DR7
  – LL_RTC_BKP_DR8
  – LL_RTC_BKP_DR9
  – LL_RTC_BKP_DR10
  – LL_RTC_BKP_DR11
  – LL_RTC_BKP_DR12
  – LL_RTC_BKP_DR13
  – LL_RTC_BKP_DR14
  – LL_RTC_BKP_DR15
  – LL_RTC_BKP_DR16
  – LL_RTC_BKP_DR17
  – LL_RTC_BKP_DR18
  – LL_RTC_BKP_DR19
  – LL_RTC_BKP_DR20
  – LL_RTC_BKP_DR21
  – LL_RTC_BKP_DR22
  – LL_RTC_BKP_DR23
  – LL_RTC_BKP_DR24
  – LL_RTC_BKP_DR25
  – LL_RTC_BKP_DR26
  – LL_RTC_BKP_DR27
  – LL_RTC_BKP_DR28
  – LL_RTC_BKP_DR29
  – LL_RTC_BKP_DR30
  – LL_RTC_BKP_DR31
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_SetRegister

LL_RTC_BKP_GetRegister

Function name

__STATIC_INLINE uint32_t LL_RTC_BKP_GetRegister (const RTC_TypeDef * RTCx, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11
  - LL_RTC_BKP_DR12
  - LL_RTC_BKP_DR13
  - LL_RTC_BKP_DR14
  - LL_RTC_BKP_DR15
  - LL_RTC_BKP_DR16
  - LL_RTC_BKP_DR17
  - LL_RTC_BKP_DR18
  - LL_RTC_BKP_DR19
  - LL_RTC_BKP_DR20
  - LL_RTC_BKP_DR21
  - LL_RTC_BKP_DR22
  - LL_RTC_BKP_DR23
  - LL_RTC_BKP_DR24
  - LL_RTC_BKP_DR25
  - LL_RTC_BKP_DR26
  - LL_RTC_BKP_DR27
  - LL_RTC_BKP_DR28
  - LL_RTC_BKP_DR29
  - LL_RTC_BKP_DR30
  - LL_RTC_BKP_DR31

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Reference Manual to LL API cross reference:

- TAMP_BKPxR BKP LL_RTC_BKP_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name

__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)

Function description

Set Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance
- **Frequency:** This parameter can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Return values

- **None:**

Notes

- Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_SetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (const RTC_TypeDef * RTCx)

Function description

Get Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Reference Manual to LL API cross reference:

- RTC_CR COE LL_RTC_CAL_GetOutputFreq
- RTC_CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)

Function description

Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance
- **Pulse:** This parameter can be one of the following values:
  - LL_RTC_CALIB_INSERTPULSE_NONE
  - LL_RTC_CALIB_INSERTPULSE_SET

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (const RTC_TypeDef * RTCx)

Function description

Check if one RTCCLK has been inserted or not every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR CALP LL_RTC_CAL_IsPulseInserted

LL_RTC_CAL_SetPeriod

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPeriod (RTC_TypeDef * RTCx, uint32_t Period)

Function description

Set the calibration cycle period.

Parameters

- **RTCx:** RTC Instance
- **Period:** This parameter can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_SetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (const RTC_TypeDef * RTCx)

Function description

Get the calibration cycle period.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Reference Manual to LL API cross reference:

- RTC_CALR CALW8 LL_RTC_CAL_GetPeriod
- RTC_CALR CALW16 LL_RTC_CAL_GetPeriod

LL_RTC_CAL_SetMinus

Function name

__STATIC_INLINE void LL_RTC_CAL_SetMinus (RTC_TypeDef * RTCx, uint32_t CalibMinus)

Function description

Set Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ICSR

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (const RTC_TypeDef * RTCx)

Function description

Get Calibration minus.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data= 0x1FF

Reference Manual to LL API cross reference:

- RTC_CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_CAL_LowPower_Enable

Function name

__STATIC_INLINE void LL_RTC_CAL_LowPower_Enable (RTC_TypeDef * RTCx)

Function description

Enable Calibration Low Power.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_Enable

LL_RTC_CAL_LowPower_Disable

Function name

__STATIC_INLINE void LL_RTC_CAL_LowPower_Disable (RTC_TypeDef * RTCx)

Function description

Disable Calibration Low Power.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_Disable

LL_RTC_CAL_LowPower_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_LowPower_IsEnabled (const RTC_TypeDef * RTCx)

Function description

Check if Calibration Low Power is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CALR LPCAL LL_RTC_CAL_LowPower_IsEnabled

LL_RTC_IsActiveFlag_ITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITS (const RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ITSF LL_RTC_IsActiveFlag_ITS

LL_RTC_IsActiveFlag_RECALP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (const RTC_TypeDef * RTCx)

Function description

Get Recalibration pending Flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_TSOV

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOV (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSOVF LL_RTC_IsActiveFlag_TSOV

LL_RTC_IsActiveFlag_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TS (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR TSF LL_RTC_IsActiveFlag_TS

LL_RTC_IsActiveFlag_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRB (const RTC_TypeDef * RTCx)

Function description

Get Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRBF LL_RTC_IsActiveFlag_ALRB

LL_RTC_IsActiveFlag_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (const RTC_TypeDef * RTCx)

Function description

Get Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_IsActiveFlag_SSRU

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SSRU (const RTC_TypeDef * RTCx)

Function description

Get SSR Underflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_SR SSRUF LL_RTC_IsActiveFlag_SSRU

LL_RTC_ClearFlag_ITS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITS (RTC_TypeDef * RTCx)

Function description

Clear Internal Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CITSF LL_RTC_ClearFlag_ITS

LL_RTC_ClearFlag_TSOV

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TSOV (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp overflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSOVF LL_RTC_ClearFlag_TSOV

LL_RTC_ClearFlag_TS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TS (RTC_TypeDef * RTCx)

Function description

Clear Time-stamp flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CTSF LL_RTC_ClearFlag_TS

LL_RTC_ClearFlag_WUT

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)

Function description

Clear Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CWUTF LL_RTC_ClearFlag_WUT

LL_RTC_ClearFlag_ALRB

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRB (RTC_TypeDef * RTCx)

Function description

Clear Alarm B flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRBF LL_RTC_ClearFlag_ALRB

LL_RTC_ClearFlag_ALRA

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ALRA (RTC_TypeDef * RTCx)

Function description

Clear Alarm A flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_ClearFlag_SSRU

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_SSRU (RTC_TypeDef * RTCx)

Function description

Clear SSR Underflow flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_SCR CSSRUF LL_RTC_ClearFlag_SSRU

LL_RTC_IsActiveFlag_INIT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (const RTC_TypeDef * RTCx)

Function description

Get Initialization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (const RTC_TypeDef * RTCx)

Function description

Get Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_IsActiveFlag_RS

LL_RTC_ClearFlag_RS

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_RS (RTC_TypeDef * RTCx)

Function description

Clear Registers synchronization flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTC_ICSR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (const RTC_TypeDef * RTCx)

Function description

Get Initialization status flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (const RTC_TypeDef * RTCx)

Function description

Get Shift operation pending flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRAM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAM (const RTC_TypeDef * RTCx)

Function description

Get Alarm A masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRAMF LL_RTC_IsActiveFlag_ALRAM

LL_RTC_IsActiveFlag_SSRUM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SSRUM (const RTC_TypeDef * RTCx)

Function description

Get SSR Underflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR SSRUMF LL_RTC_IsActiveFlag_SSRUM

LL_RTC_IsActiveFlag_ALRBM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRBM (const RTC_TypeDef * RTCx)

Function description

Get Alarm B masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ALRBMF LL_RTC_IsActiveFlag_ALRBM

LL_RTC_IsActiveFlag_WUTM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTM (const RTC_TypeDef * RTCx)

Function description

Get Wakeup timer masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR WUTMF LL_RTC_IsActiveFlag_WUTM

LL_RTC_IsActiveFlag_TSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSM (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSMF LL_RTC_IsActiveFlag_TSM

LL_RTC_IsActiveFlag_TSOVM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TSOVM (const RTC_TypeDef * RTCx)

Function description

Get Time-stamp overflow masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR TSOVMF LL_RTC_IsActiveFlag_TSOVM

LL_RTC_IsActiveFlag_ITSM

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITSM (const RTC_TypeDef * RTCx)

Function description

Get Internal Time-stamp masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_MISR ITSMF LL_RTC_IsActiveFlag_ITSM

LL_RTC_IsActiveFlag_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Get tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP1F LL_RTC_IsActiveFlag_TAMP1

LL_RTC_IsActiveFlag_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Get tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP2F LL_RTC_IsActiveFlag_TAMP2

LL_RTC_IsActiveFlag_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Get tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP3F LL_RTC_IsActiveFlag_TAMP3

LL_RTC_IsActiveFlag_TAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Get tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP4F LL_RTC_IsActiveFlag_TAMP4

LL_RTC_IsActiveFlag_TAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Get tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP5F LL_RTC_IsActiveFlag_TAMP5

LL_RTC_IsActiveFlag_TAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Get tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP6F LL_RTC_IsActiveFlag_TAMP6

LL_RTC_IsActiveFlag_TAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Get tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP7F LL_RTC_IsActiveFlag_TAMP7

LL_RTC_IsActiveFlag_TAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Get tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR TAMP8F LL_RTC_IsActiveFlag_TAMP8

LL_RTC_IsActiveFlag_ITAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP1F LL_RTC_IsActiveFlag_ITAMP1

LL_RTC_IsActiveFlag_ITAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP2F LL_RTC_IsActiveFlag_ITAMP2

LL_RTC_IsActiveFlag_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP3F LL_RTC_IsActiveFlag_ITAMP3

LL_RTC_IsActiveFlag_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP5F LL_RTC_IsActiveFlag_ITAMP5

LL_RTC_IsActiveFlag_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP6F LL_RTC_IsActiveFlag_ITAMP6

LL_RTC_IsActiveFlag_ITAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP7F LL_RTC_IsActiveFlag_ITAMP7

LL_RTC_IsActiveFlag_ITAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP8F LL_RTC_IsActiveFlag_ITAMP8

LL_RTC_IsActiveFlag_ITAMP9

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 9 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP9F LL_RTC_IsActiveFlag_ITAMP9

LL_RTC_IsActiveFlag_ITAMP11

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 11 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP11F LL_RTC_IsActiveFlag_ITAMP11

LL_RTC_IsActiveFlag_ITAMP12

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 12 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP12F LL_RTC_IsActiveFlag_ITAMP12

LL_RTC_IsActiveFlag_ITAMP13

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 13 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_SR ITAMP13F LL_RTC_IsActiveFlag_ITAMP13

LL_RTC_IsActiveFlag_TAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP1M (const RTC_TypeDef * RTCx)

Function description

Get tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP1MF LL_RTC_IsActiveFlag_TAMP1M

LL_RTC_IsActiveFlag_TAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP2M (const RTC_TypeDef * RTCx)

Function description

Get tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP2MF LL_RTC_IsActiveFlag_TAMP2M

LL_RTC_IsActiveFlag_TAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP3M (const RTC_TypeDef * RTCx)

Function description

Get tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP3MF LL_RTC_IsActiveFlag_TAMP3M

LL_RTC_IsActiveFlag_TAMP4M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP4M (const RTC_TypeDef * RTCx)

Function description

Get tamper 4 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP4MF LL_RTC_IsActiveFlag_TAMP4M

LL_RTC_IsActiveFlag_TAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP5M (const RTC_TypeDef * RTCx)

Function description

Get tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP5MF LL_RTC_IsActiveFlag_TAMP5M

LL_RTC_IsActiveFlag_TAMP6M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP6M (const RTC_TypeDef * RTCx)

Function description

Get tamper 6 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP3MF LL_RTC_IsActiveFlag_TAMP6M

LL_RTC_IsActiveFlag_TAMP7M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP7M (const RTC_TypeDef * RTCx)

Function description

Get tamper 7 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP7MF LL_RTC_IsActiveFlag_TAMP7M

LL_RTC_IsActiveFlag_TAMP8M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_TAMP8M (const RTC_TypeDef * RTCx)

Function description

Get tamper 8 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR TAMP8MF LL_RTC_IsActiveFlag_TAMP8M

LL_RTC_IsActiveFlag_ITAMP1M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP1M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 1 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP1MF LL_RTC_IsActiveFlag_ITAMP1M

LL_RTC_IsActiveFlag_ITAMP2M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP2M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 2 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP2MF LL_RTC_IsActiveFlag_ITAMP2M

LL_RTC_IsActiveFlag_ITAMP3M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP3M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 3 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP3MF LL_RTC_IsActiveFlag_ITAMP3M

LL_RTC_IsActiveFlag_ITAMP5M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP5M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 5 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP5MF LL_RTC_IsActiveFlag_ITAMP5M

LL_RTC_IsActiveFlag_ITAMP8M

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ITAMP8M (const RTC_TypeDef * RTCx)

Function description

Get internal tamper 8 interrupt masked flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_MISR ITAMP8MF LL_RTC_IsActiveFlag_ITAMP8M

LL_RTC_ClearFlag_TAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP1F LL_RTC_ClearFlag_TAMP1

LL_RTC_ClearFlag_TAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP2F LL_RTC_ClearFlag_TAMP2

LL_RTC_ClearFlag_TAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP3F LL_RTC_ClearFlag_TAMP3

LL_RTC_ClearFlag_TAMP4

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 4 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP3F LL_RTC_ClearFlag_TAMP4

LL_RTC_ClearFlag_TAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP5F LL_RTC_ClearFlag_TAMP5

LL_RTC_ClearFlag_TAMP6

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 6 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP6F LL_RTC_ClearFlag_TAMP6

LL_RTC_ClearFlag_TAMP7

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 7 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP7F LL_RTC_ClearFlag_TAMP7

LL_RTC_ClearFlag_TAMP8

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Clear tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CTAMP8F LL_RTC_ClearFlag_TAMP8

LL_RTC_ClearFlag_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 1 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP1F LL_RTC_ClearFlag_ITAMP1

LL_RTC_ClearFlag_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 2 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP2F LL_RTC_ClearFlag_ITAMP2

LL_RTC_ClearFlag_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 3 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP3F LL_RTC_ClearFlag_ITAMP3

LL_RTC_ClearFlag_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 5 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP5F LL_RTC_ClearFlag_ITAMP5

LL_RTC_ClearFlag_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Clear internal tamper 8 detection flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SCR CITAMP8F LL_RTC_ClearFlag_ITAMP8

LL_RTC_SetRtcSecure

Function name

__STATIC_INLINE void LL_RTC_SetRtcSecure (RTC_TypeDef * RTCx, uint32_t rtcSecure)

Function description

Set RTC secure level.

Parameters

- **RTCx:** RTC Instance
- **rtcSecure:** This parameter can be a combination of the following values:
  – LL_RTC_SECURE_FULL_YES
  – LL_RTC_SECURE_FULL_NO
  – LL_RTC_UNSECURE_FEATURE_INIT
  – LL_RTC_UNSECURE_FEATURE_CAL
  – LL_RTC_UNSECURE_FEATURE_TS
  – LL_RTC_UNSECURE_FEATURE_WUT
  – LL_RTC_UNSECURE_FEATURE_ALRA
  – LL_RTC_UNSECURE_FEATURE_ALRB

Return values

- **None:**

Notes

- secure features are relevant if LL_RTC_SECURE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_SECCFGR SEC LL_RTC_SetRtcSecure
- RTC_SECCFGR INITSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR CALSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR TSSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR WUTSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRASEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRBSEC LL_RTC_SetRtcSecure

LL_RTC_GetRtcSecure

Function name

__STATIC_INLINE uint32_t LL_RTC_GetRtcSecure (const RTC_TypeDef * RTCx)

Function description

Get RTC secure level.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of the following values:
  – LL_RTC_SECURE_FULL_YES
  – LL_RTC_SECURE_FULL_NO
  – LL_RTC_UNSECURE_FEATURE_INIT
  – LL_RTC_UNSECURE_FEATURE_CAL
  – LL_RTC_UNSECURE_FEATURE_TS
  – LL_RTC_UNSECURE_FEATURE_WUT
  – LL_RTC_UNSECURE_FEATURE_ALRA
  – LL_RTC_UNSECURE_FEATURE_ALRB

Notes

- Secure features is relevant if LL_RTC_SECURE_FULL_YES.

Reference Manual to LL API cross reference:

- RTC_SECCFGR SEC LL_RTC_SetRtcSecure
- RTC_SECCFGR INISEC LL_RTC_SetRtcSecure
- RTC_SECCFGR CALSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR TSSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR WUTSEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRASEC LL_RTC_SetRtcSecure
- RTC_SECCFGR ALRBSEC LL_RTC_SetRtcSecure

LL_RTC_SetTampSecure

Function name

__STATIC_INLINE void LL_RTC_SetTampSecure (const RTC_TypeDef * RTCx, uint32_t tampSecure)

Function description

Set TAMPER secure level.

Parameters

- **RTCx:** RTC Instance
- **tampSecure:** This parameter can be one of the following values:
  – LL_TAMP_SECURE_FULL_YES
  – LL_TAMP_SECURE_FULL_NO

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_SetTampSecure

LL_RTC_GetTampSecure

Function name

__STATIC_INLINE uint32_t LL_RTC_GetTampSecure (const RTC_TypeDef * RTCx)

Function description

Get TAMPER secure level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  – LL_TAMP_SECURE_FULL_YES
  – LL_TAMP_SECURE_FULL_NO

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_GetTampSecure

LL_RTC_SetRtcPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetRtcPrivilege (RTC_TypeDef * RTCx, uint32_t rtcPrivilege)

Function description

Set RTC privilege level.

Parameters

- **RTCx:** RTC Instance
- **rtcPrivilege:** This parameter can be a combination of the following values:
  - LL_RTC_PRIVILEGE_FULL_YES
  - LL_RTC_PRIVILEGE_FULL_NO
  - LL_RTC_PRIVILEGE_FEATURE_INIT
  - LL_RTC_PRIVILEGE_FEATURE_CAL
  - LL_RTC_PRIVILEGE_FEATURE_TS
  - LL_RTC_PRIVILEGE_FEATURE_WUT
  - LL_RTC_PRIVILEGE_FEATURE_ALRA
  - LL_RTC_PRIVILEGE_FEATURE_ALRB

Return values

- **None:**

Notes

- Privilege features are relevant if LL_RTC_PRIVILEGE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_PRIVCFGR PRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR INITPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR CALPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR TSPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR WUTPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRAPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRBPRIV LL_RTC_SetRtcPrivilege

LL_RTC_GetRtcPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetRtcPrivilege (const RTC_TypeDef * RTCx)

Function description

Get RTC privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of the following values:
  - LL_RTC_PRIVILEGE_FULL_YES
  - LL_RTC_PRIVILEGE_FULL_NO
  - LL_RTC_PRIVILEGE_FEATURE_INIT
  - LL_RTC_PRIVILEGE_FEATURE_CAL
  - LL_RTC_PRIVILEGE_FEATURE_TS
  - LL_RTC_PRIVILEGE_FEATURE_WUT
  - LL_RTC_PRIVILEGE_FEATURE_ALRA
  - LL_RTC_PRIVILEGE_FEATURE_ALRB

Notes

- Privilege features are relevant if LL_RTC_PRIVILEGE_FULL_NO.

Reference Manual to LL API cross reference:

- RTC_PRIVCFGR PRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR INITPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR CALPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR TSPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR WUTPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRAPRIV LL_RTC_SetRtcPrivilege
- RTC_PRIVCFGR ALRBPRIV LL_RTC_SetRtcPrivilege

LL_RTC_SetTampPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetTampPrivilege (const RTC_TypeDef * RTCx, uint32_t tampPrivilege)

Function description

Set TAMPER privilege level.

Parameters

- **RTCx:** RTC Instance
- **tampPrivilege:** This parameter can be one of the following values:
  – LL_TAMP_PRIVILEGE_FULL_YES
  – LL_TAMP_PRIVILEGE_FULL_NO

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPPRIV LL_RTC_SetTampPrivilege

LL_RTC_GetTampPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetTampPrivilege (const RTC_TypeDef * RTCx)

Function description

Get TAMPER privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  – LL_TAMP_PRIVILEGE_FULL_YES
  – LL_TAMP_PRIVILEGE_FULL_NO

Reference Manual to LL API cross reference:

- TAMP_SECCFGR TAMPSEC LL_RTC_GetTampPrivilege

LL_RTC_SetBackupRegisterPrivilege

Function name

__STATIC_INLINE void LL_RTC_SetBackupRegisterPrivilege (const RTC_TypeDef * RTCx, uint32_t bckupRegisterPrivilege)

Function description

Set Backup Registers privilege level.

Parameters

- **RTCx:** RTC Instance
- **bckupRegisterPrivilege:** This parameter can be one of the following values:
  – LL_RTC_PRIVILEGE_BKUP_ZONE_NONE
  – LL_RTC_PRIVILEGE_BKUP_ZONE_1
  – LL_RTC_PRIVILEGE_BKUP_ZONE_2
  – LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Return values

- **None:**

Notes

- bckupRegisterPrivilege is only writable in secure mode or if trustzone is disabled

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR BKPWPRIV LL_RTC_SetTampPrivilege
- TAMP_PRIVCFGR BKPRWPRIV LL_RTC_SetTampPrivilege

LL_RTC_GetBackupRegisterPrivilege

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegisterPrivilege (const RTC_TypeDef * RTCx)

Function description

Get Backup Registers privilege level.

Parameters

- **RTCx:** RTC Instance

Return values

- **This:** parameter can be one of the following values:
  – LL_RTC_PRIVILEGE_BKUP_ZONE_NONE
  – LL_RTC_PRIVILEGE_BKUP_ZONE_1
  – LL_RTC_PRIVILEGE_BKUP_ZONE_2
  – LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Reference Manual to LL API cross reference:

- TAMP_PRIVCFGR BKPWPRIV LL_RTC_SetTampPrivilege
- TAMP_PRIVCFGR BKPRWPRIV LL_RTC_SetTampPrivilege

LL_RTC_SetBackupRegProtection

Function name

__STATIC_INLINE void LL_RTC_SetBackupRegProtection (const RTC_TypeDef * RTCx, uint32_t startZone2, uint32_t startZone3)

Function description

Set Backup registers protection level.

Parameters

- **RTCx:** RTC Instance
- **startZone2:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11
  - LL_RTC_BKP_DR12
  - LL_RTC_BKP_DR13
  - LL_RTC_BKP_DR14
  - LL_RTC_BKP_DR15
  - LL_RTC_BKP_DR16
  - LL_RTC_BKP_DR17
  - LL_RTC_BKP_DR18
  - LL_RTC_BKP_DR19
  - LL_RTC_BKP_DR20
  - LL_RTC_BKP_DR21
  - LL_RTC_BKP_DR22
  - LL_RTC_BKP_DR23
  - LL_RTC_BKP_DR24
  - LL_RTC_BKP_DR25
  - LL_RTC_BKP_DR26
  - LL_RTC_BKP_DR27
  - LL_RTC_BKP_DR28
  - LL_RTC_BKP_DR29
  - LL_RTC_BKP_DR30
  - LL_RTC_BKP_DR31

- **startZone3:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
  - LL_RTC_BKP_DR5
  - LL_RTC_BKP_DR6
  - LL_RTC_BKP_DR7
  - LL_RTC_BKP_DR8
  - LL_RTC_BKP_DR9
  - LL_RTC_BKP_DR10
  - LL_RTC_BKP_DR11
  - LL_RTC_BKP_DR12
  - LL_RTC_BKP_DR13
  - LL_RTC_BKP_DR14
  - LL_RTC_BKP_DR15
  - LL_RTC_BKP_DR16
  - LL_RTC_BKP_DR17
  - LL_RTC_BKP_DR18
  - LL_RTC_BKP_DR19
  - LL_RTC_BKP_DR20
  - LL_RTC_BKP_DR21
  - LL_RTC_BKP_DR22
  - LL_RTC_BKP_DR23
  - LL_RTC_BKP_DR24
  - LL_RTC_BKP_DR25
  - LL_RTC_BKP_DR26
  - LL_RTC_BKP_DR27
  - LL_RTC_BKP_DR28
  - LL_RTC_BKP_DR29
  - LL_RTC_BKP_DR30
  - LL_RTC_BKP_DR31

Return values

- **None:**

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection write protection
- Zone 3 : read non-protection write non-protection
- zone 1 : start from 0 to startZone2 start value
- zone 2 : start from startZone2 start value to startZone3 start value
- zone 3 : start from to startZone3 to the end of BACKUPREG
- Warning : this parameter is only writable in secure mode or if trustzone is disabled

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPWSEC LL_RTC_SetBackupRegProtection
- TAMP_SECCFGR BKPRWSEC LL_RTC_SetBackupRegProtection

LL_RTC_GetBackupRegProtectionStartZone2

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegProtectionStartZone2 (const RTC_TypeDef * RTCx)

Function description

Get Backup registers protection level start zone 2.

Parameters

- **RTCx:** RTC Instance

Return values

- **Start:** zone 2

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection/non-privile write protection
- Zone 3 : read non-protection write non-protection

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPRWSEC LL_RTC_GetBackupRegProtectionStartZone2

LL_RTC_GetBackupRegProtectionStartZone3

Function name

__STATIC_INLINE uint32_t LL_RTC_GetBackupRegProtectionStartZone3 (const RTC_TypeDef * RTCx)

Function description

Get Backup registers protection level start zone 3.

Parameters

- **RTCx:** RTC Instance

Return values

- **Start:** zone 2

Notes

- Zone 1 : read protection write protection
- Zone 2 : read non-protection write protection
- Zone 3 : read non-protection write non-protection

Reference Manual to LL API cross reference:

- TAMP_SECCFGR BKPWSEC LL_RTC_GetBackupRegProtectionStartZone3

LL_RTC_EnableIT_TS

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TS (RTC_TypeDef * RTCx)

Function description

Enable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_EnableIT_TS

LL_RTC_DisableIT_TS

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TS (RTC_TypeDef * RTCx)

Function description

Disable Time-stamp interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_DisableIT_TS

LL_RTC_EnableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_DisableIT_WUT

LL_RTC_EnableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Enable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_EnableIT_ALRB

LL_RTC_DisableIT_ALRB

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRB (RTC_TypeDef * RTCx)

Function description

Disable Alarm B interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_DisableIT_ALRB

LL_RTC_EnableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Enable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_EnableIT_ALRA

LL_RTC_DisableIT_ALRA

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ALRA (RTC_TypeDef * RTCx)

Function description

Disable Alarm A interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_EnableIT_SSRU

Function name

__STATIC_INLINE void LL_RTC_EnableIT_SSRU (RTC_TypeDef * RTCx)

Function description

Enable SSR Underflow interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_EnableIT_SSRU

LL_RTC_DisableIT_SSRU

Function name

__STATIC_INLINE void LL_RTC_DisableIT_SSRU (RTC_TypeDef * RTCx)

Function description

Disable SSR Underflow interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_DisableIT_SSRU

LL_RTC_IsEnabledIT_TS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TS (const RTC_TypeDef * RTCx)

Function description

Check if Time-stamp interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR TSIE LL_RTC_IsEnabledIT_TS

LL_RTC_IsEnabledIT_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (const RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRB

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRB (const RTC_TypeDef * RTCx)

Function description

Check if Alarm B interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRBIE LL_RTC_IsEnabledIT_ALRB

LL_RTC_IsEnabledIT_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (const RTC_TypeDef * RTCx)

Function description

Check if Alarm A interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR ALRAIE LL_RTC_IsEnabledIT_ALRA

LL_RTC_IsEnabledIT_SSRU

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_SSRU (const RTC_TypeDef * RTCx)

Function description

Check if SSR Underflow interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_CR SSRUIE LL_RTC_IsEnabledIT_SSRU

LL_RTC_EnableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_EnableIT_TAMP1

LL_RTC_DisableIT_TAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_DisableIT_TAMP1

LL_RTC_EnableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_EnableIT_TAMP2

LL_RTC_DisableIT_TAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_DisableIT_TAMP2

LL_RTC_EnableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_EnableIT_TAMP3

LL_RTC_DisableIT_TAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_DisableIT_TAMP3

LL_RTC_EnableIT_TAMP4

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_EnableIT_TAMP4

LL_RTC_DisableIT_TAMP4

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 4 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_DisableIT_TAMP4

LL_RTC_EnableIT_TAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_EnableIT_TAMP5

LL_RTC_DisableIT_TAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP5IE LL_RTC_DisableIT_TAMP5

LL_RTC_EnableIT_TAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_EnableIT_TAMP6

LL_RTC_DisableIT_TAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_DisableIT_TAMP6

LL_RTC_EnableIT_TAMP7

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_EnableIT_TAMP7

LL_RTC_DisableIT_TAMP7

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP7IE LL_RTC_DisableIT_TAMP7

LL_RTC_EnableIT_TAMP8

Function name

__STATIC_INLINE void LL_RTC_EnableIT_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Enable tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_EnableIT_TAMP8

LL_RTC_DisableIT_TAMP8

Function name

__STATIC_INLINE void LL_RTC_DisableIT_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Disable tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_DisableIT_TAMP8

LL_RTC_EnableIT_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_EnableIT_ITAMP1

LL_RTC_DisableIT_ITAMP1

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 1 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_DisableIT_ITAMP1

LL_RTC_EnableIT_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_EnableIT_ITAMP2

LL_RTC_DisableIT_ITAMP2

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 2 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_DisableIT_ITAMP2

LL_RTC_EnableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_EnableIT_ITAMP3

LL_RTC_DisableIT_ITAMP3

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 3 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_DisableIT_ITAMP3

LL_RTC_EnableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_EnableIT_ITAMP5

LL_RTC_DisableIT_ITAMP5

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 5 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_DisableIT_ITAMP5

LL_RTC_EnableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_EnableIT_ITAMP6

LL_RTC_DisableIT_ITAMP6

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 6 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_DisableIT_ITAMP6

LL_RTC_EnableIT_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_EnableIT_ITAMP7

LL_RTC_DisableIT_ITAMP7

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 7 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_DisableIT_ITAMP7

LL_RTC_EnableIT_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_EnableIT_ITAMP8

LL_RTC_DisableIT_ITAMP8

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 8 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_DisableIT_ITAMP8

LL_RTC_EnableIT_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 9 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_EnableIT_ITAMP9

LL_RTC_DisableIT_ITAMP9

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 9 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_DisableIT_ITAMP9

LL_RTC_EnableIT_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 11 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_EnableIT_ITAMP11

LL_RTC_DisableIT_ITAMP11

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 11 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_DisableIT_ITAMP11

LL_RTC_EnableIT_ITAMP12

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 12 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP12IE LL_RTC_EnableIT_ITAMP12

LL_RTC_DisableIT_ITAMP12

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 12 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP12IE LL_RTC_DisableIT_ITAMP12

LL_RTC_EnableIT_ITAMP13

Function name

__STATIC_INLINE void LL_RTC_EnableIT_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Enable internal tamper 13 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP13IE LL_RTC_EnableIT_ITAMP13

LL_RTC_DisableIT_ITAMP13

Function name

__STATIC_INLINE void LL_RTC_DisableIT_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Disable internal tamper 13 interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TAMP_IER TAMP13IE LL_RTC_DisableIT_ITAMP13

LL_RTC_IsEnabledIT_TAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP1 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_IsEnabledIT_TAMP1

LL_RTC_IsEnabledIT_TAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP2 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP2IE LL_RTC_IsEnabledIT_TAMP2

LL_RTC_IsEnabledIT_TAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP3 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP3IE LL_RTC_IsEnabledIT_TAMP3

LL_RTC_IsEnabledIT_TAMP4

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP4 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 4 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP4IE LL_RTC_IsEnabledIT_TAMP4

LL_RTC_IsEnabledIT_TAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP5 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_IsEnabledIT_TAMP5

LL_RTC_IsEnabledIT_TAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP6 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP6IE LL_RTC_IsEnabledIT_TAMP6

LL_RTC_IsEnabledIT_TAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP7 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 7 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP1IE LL_RTC_IsEnabledIT_TAMP7

LL_RTC_IsEnabledIT_TAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_TAMP8 (const RTC_TypeDef * RTCx)

Function description

Check if tamper 8 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER TAMP8IE LL_RTC_IsEnabledIT_TAMP8

LL_RTC_IsEnabledIT_ITAMP1

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP1 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 1 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP1IE LL_RTC_IsEnabledIT_ITAMP1

LL_RTC_IsEnabledIT_ITAMP2

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP2 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 2 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP2IE LL_RTC_IsEnabledIT_ITAMP2

LL_RTC_IsEnabledIT_ITAMP3

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP3 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 3 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP3IE LL_RTC_IsEnabledIT_ITAMP3

LL_RTC_IsEnabledIT_ITAMP5

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP5 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 5 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP5IE LL_RTC_IsEnabledIT_ITAMP5

LL_RTC_IsEnabledIT_ITAMP6

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP6 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 6 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP6IE LL_RTC_IsEnabledIT_ITAMP6

LL_RTC_IsEnabledIT_ITAMP7

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP7 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 7 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP7IE LL_RTC_IsEnabledIT_ITAMP7

LL_RTC_IsEnabledIT_ITAMP8

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP8 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 8 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP8IE LL_RTC_IsEnabledIT_ITAMP8

LL_RTC_IsEnabledIT_ITAMP9

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP9 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 9 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP9IE LL_RTC_IsEnabledIT_ITAMP9

LL_RTC_IsEnabledIT_ITAMP11

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP11 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 11 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP11IE LL_RTC_IsEnabledIT_ITAMP11

LL_RTC_IsEnabledIT_ITAMP12

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP12 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 12 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP12IE LL_RTC_IsEnabledIT_ITAMP12

LL_RTC_IsEnabledIT_ITAMP13

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ITAMP13 (const RTC_TypeDef * RTCx)

Function description

Check if internal tamper 13 interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- TAMP_IER ITAMP13IE LL_RTC_IsEnabledIT_ITAMP13

LL_RTC_IncrementMonotonicCounter

Function name

__STATIC_INLINE void LL_RTC_IncrementMonotonicCounter (const RTC_TypeDef * RTCx)

Function description

Increment Monotonic counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TAMP_COUNTR COUNT LL_RTC_IncrementMonotonicCounter

LL_RTC_GetMonotonicCounter

Function name

__STATIC_INLINE uint32_t LL_RTC_GetMonotonicCounter (const RTC_TypeDef * RTCx)

Function description

Increment Monotonic counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **Monotonic:** counter value.

Reference Manual to LL API cross reference:

- TAMP_COUNTR COUNT LL_RTC_GetMonotonicCounter

LL_RTC_DeInit

Function name

ErrorStatus LL_RTC_DeInit (RTC_TypeDef * RTCx)

Function description

De-Initializes the RTC registers to their default reset values.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC registers are de-initialized
  – ERROR: RTC registers are not de-initialized

Notes

- This function does not reset the RTC Clock source and RTC Backup Data registers.

LL_RTC_Init

Function name

ErrorStatus LL_RTC_Init (RTC_TypeDef * RTCx, LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Initializes the RTC registers according to the specified parameters in RTC_InitStruct.

Parameters

- **RTCx:** RTC Instance
- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure that contains the configuration information for the RTC peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC registers are initialized
  – ERROR: RTC registers are not initialized

Notes

- The RTC Prescaler register is write protected and can be written in initialization mode only.

LL_RTC_StructInit

Function name

void LL_RTC_StructInit (LL_RTC_InitTypeDef * RTC_InitStruct)

Function description

Set each LL_RTC_InitTypeDef field to default value.

Parameters

- **RTC_InitStruct:** pointer to a LL_RTC_InitTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_TIME_Init

Function name

ErrorStatus LL_RTC_TIME_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set the RTC current time.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  – LL_RTC_FORMAT_BIN
  – LL_RTC_FORMAT_BCD
- **RTC_TimeStruct:** pointer to a RTC_TimeTypeDef structure that contains the time configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC Time register is configured
  – ERROR: RTC Time register is not configured

LL_RTC_TIME_StructInit

Function name

void LL_RTC_TIME_StructInit (LL_RTC_TimeTypeDef * RTC_TimeStruct)

Function description

Set each LL_RTC_TimeTypeDef field to default value (Time = 00h:00min:00sec).

Parameters

- **RTC_TimeStruct:** pointer to a LL_RTC_TimeTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_DATE_Init

Function name

ErrorStatus LL_RTC_DATE_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set the RTC current date.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_DateStruct:** pointer to a RTC_DateTypeDef structure that contains the date configuration information for the RTC.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC Day register is configured
  - ERROR: RTC Day register is not configured

LL_RTC_DATE_StructInit

Function name

void LL_RTC_DATE_StructInit (LL_RTC_DateTypeDef * RTC_DateStruct)

Function description

Set each LL_RTC_DateTypeDef field to default value (date = Monday, January 01 xx00)

Parameters

- **RTC_DateStruct:** pointer to a LL_RTC_DateTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMA_Init

Function name

ErrorStatus LL_RTC_ALMA_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm A.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  - LL_RTC_FORMAT_BIN
  - LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: ALARMA registers are configured
  – ERROR: ALARMA registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use LL_RTC_ALMA_Disable function).

LL_RTC_ALMB_Init

Function name

ErrorStatus LL_RTC_ALMB_Init (RTC_TypeDef * RTCx, uint32_t RTC_Format, LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set the RTC Alarm B.

Parameters

- **RTCx:** RTC Instance
- **RTC_Format:** This parameter can be one of the following values:
  – LL_RTC_FORMAT_BIN
  – LL_RTC_FORMAT_BCD
- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure that contains the alarm configuration parameters.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: ALARMB registers are configured
  – ERROR: ALARMB registers are not configured

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (LL_RTC_ALMB_Disable function).

LL_RTC_ALMA_StructInit

Function name

void LL_RTC_ALMA_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_ALMB_StructInit

Function name

void LL_RTC_ALMB_StructInit (LL_RTC_AlarmTypeDef * RTC_AlarmStruct)

Function description

Set each LL_RTC_AlarmTypeDef of ALARMA field to default value (Time = 00h:00mn:00sec / Day = 1st day of the month/Mask = all fields are masked).

Parameters

- **RTC_AlarmStruct:** pointer to a LL_RTC_AlarmTypeDef structure which will be initialized.

Return values

- **None:**

LL_RTC_EnterInitMode

Function name

ErrorStatus LL_RTC_EnterInitMode (RTC_TypeDef * RTCx)

Function description

Enters the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC is in Init mode
  - ERROR: RTC is not in Init mode

Notes

- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_ExitInitMode

Function name

ErrorStatus LL_RTC_ExitInitMode (RTC_TypeDef * RTCx)

Function description

Exit the RTC Initialization mode.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: RTC exited from in Init mode
  - ERROR: Not applicable

Notes

- When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
- The RTC Initialization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.

LL_RTC_WaitForSynchro

Function name

ErrorStatus LL_RTC_WaitForSynchro (RTC_TypeDef * RTCx)

Function description

Waits until the RTC Time and Day registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: RTC registers are synchronised
  – ERROR: RTC registers are not synchronised

Notes

- The RTC Resynchronization mode is write protected, use the LL_RTC_DisableWriteProtection before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

116.3 RTC Firmware driver defines

The following section lists the various define and macros of the module.

116.3.1 RTC

RTC

ACTIVE TAMPER ASYNCHRONOUS PRESCALER CLOCK

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK

RTCCLK

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_2

RTCCLK/2

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_4

RTCCLK/4

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_8

RTCCLK/8

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_16

RTCCLK/16

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_32

RTCCLK/32

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_64

RTCCLK/64

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_128

RTCCLK/128

LL_RTC_TAMPER_ATAMP_ASYNCPRES_RTCCLK_2048

RTCCLK/2048

ACTIVE TAMPER MODE

LL_RTC_TAMPER_ATAMP_TAMP1AM
tamper 1 is active

LL_RTC_TAMPER_ATAMP_TAMP2AM
tamper 2 is active

LL_RTC_TAMPER_ATAMP_TAMP3AM
tamper 3 is active

LL_RTC_TAMPER_ATAMP_TAMP4AM
tamper 4 is active

LL_RTC_TAMPER_ATAMP_TAMP5AM
tamper 5 is active

LL_RTC_TAMPER_ATAMP_TAMP6AM
tamper 6 is active

LL_RTC_TAMPER_ATAMP_TAMP7AM
tamper 7 is active

LL_RTC_TAMPER_ATAMP_TAMP8AM
tamper 8 is active

ACTIVE TAMPER OUTPUT SELECTION

LL_RTC_TAMPER_ATAMP1IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP1IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP2IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP3IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP4IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP5IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP6IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP7IN_ATAMP8OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP1OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP2OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP3OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP4OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP5OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP6OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP7OUT

LL_RTC_TAMPER_ATAMP8IN_ATAMP8OUT

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

LL_RTC_ALARMOUT_ALMB

Alarm B output enabled

LL_RTC_ALARMOUT_WAKEUP

Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM is push-pull output

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM is open-drain output

ALARMA MASK

LL_RTC_ALMA_MASK_NONE

No masks applied on Alarm A

LL_RTC_ALMA_MASK_DATEWEEKDAY

Date/day do not care in Alarm A comparison

LL_RTC_ALMA_MASK_HOURS

Hours do not care in Alarm A comparison

LL_RTC_ALMA_MASK_MINUTES

Minutes do not care in Alarm A comparison

LL_RTC_ALMA_MASK_SECONDS

Seconds do not care in Alarm A comparison

LL_RTC_ALMA_MASK_ALL

Masks all

RTC Alarm Sub Seconds with binary mode auto clear Definitions

LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_NO

The synchronous binary counter (SS[31:0] in RTC_SSR) is free-running.

LL_RTC_ALMA_SUBSECONDBIN_AUTOCLR_YES

The synchronous binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF when reaching RTC_ALRMABINR -> SS[31:0].

ALARMA TIME FORMAT

LL_RTC_ALMA_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMA_TIME_FORMAT_PM

PM

RTC Alarm A Date WeekDay

LL_RTC_ALMA_DATEWEEKDAYSEL_DATE

Alarm A Date is selected

LL_RTC_ALMA_DATEWEEKDAYSEL_WEEKDAY

Alarm A WeekDay is selected

ALARMB MASK

LL_RTC_ALMB_MASK_NONE

No masks applied on Alarm B

LL_RTC_ALMB_MASK_DATEWEEKDAY

Date/day do not care in Alarm B comparison

LL_RTC_ALMB_MASK_HOURS

Hours do not care in Alarm B comparison

LL_RTC_ALMB_MASK_MINUTES

Minutes do not care in Alarm B comparison

LL_RTC_ALMB_MASK_SECONDS

Seconds do not care in Alarm B comparison

LL_RTC_ALMB_MASK_ALL

Masks all

Alarm Sub Seconds with binary mode auto clear Definitions

LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_NO

The synchronous binary counter (SS[31:0] in RTC_SSR) is free-running.

LL_RTC_ALMB_SUBSECONDBIN_AUTOCLR_YES

The synchronous binary counter (SS[31:0] in RTC_SSR) is running from 0xFFFF FFFF to RTC_ALRMABINR -> SS[31:0] value and is automatically reloaded with 0xFFFF FFFF when reaching RTC_ALRMABINR -> SS[31:0].

ALARMB TIME FORMAT

LL_RTC_ALMB_TIME_FORMAT_AM

AM or 24-hour format

LL_RTC_ALMB_TIME_FORMAT_PM

PM

RTC Alarm B Date WeekDay

LL_RTC_ALMB_DATEWEEKDAYSEL_DATE

Alarm B Date is selected

LL_RTC_ALMB_DATEWEEKDAYSEL_WEEKDAY

Alarm B WeekDay is selected

Calendar second incrementation in Binary mix mode

LL_RTC_BINARY_MIX_BCDU_0

1s calendar increment is generated each time SS[7:0] = 0

LL_RTC_BINARY_MIX_BCDU_1

1s calendar increment is generated each time SS[8:0] = 0

LL_RTC_BINARY_MIX_BCDU_2

1s calendar increment is generated each time SS[9:0] = 0

LL_RTC_BINARY_MIX_BCDU_3

1s calendar increment is generated each time SS[10:0] = 0

LL_RTC_BINARY_MIX_BCDU_4

1s calendar increment is generated each time SS[11:0] = 0

LL_RTC_BINARY_MIX_BCDU_5

1s calendar increment is generated each time SS[12:0] = 0

LL_RTC_BINARY_MIX_BCDU_6

1s calendar increment is generated each time SS[13:0] = 0

LL_RTC_BINARY_MIX_BCDU_7

1s calendar increment is generated each time SS[14:0] = 0

Binary mode (Sub Second Register)

LL_RTC_BINARY_NONE

Free running BCD calendar mode (Binary mode disabled).

LL_RTC_BINARY_ONLY

Free running Binary mode (BCD mode disabled)

LL_RTC_BINARY_MIX

Free running BCD calendar and Binary mode enable

BACKUP

LL_RTC_BKP_NUMBER

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

LL_RTC_BKP_DR2

LL_RTC_BKP_DR3

LL_RTC_BKP_DR4

LL_RTC_BKP_DR5

LL_RTC_BKP_DR6

LL_RTC_BKP_DR7

LL_RTC_BKP_DR8

LL_RTC_BKP_DR9

LL_RTC_BKP_DR10

LL_RTC_BKP_DR11

LL_RTC_BKP_DR12

LL_RTC_BKP_DR13

LL_RTC_BKP_DR14

LL_RTC_BKP_DR15

LL_RTC_BKP_DR16

LL_RTC_BKP_DR17

LL_RTC_BKP_DR18

LL_RTC_BKP_DR19

LL_RTC_BKP_DR20

LL_RTC_BKP_DR21

LL_RTC_BKP_DR22

LL_RTC_BKP_DR23

LL_RTC_BKP_DR24

LL_RTC_BKP_DR25

LL_RTC_BKP_DR26

LL_RTC_BKP_DR27

LL_RTC_BKP_DR28

LL_RTC_BKP_DR29

LL_RTC_BKP_DR30

LL_RTC_BKP_DR31

Calibration pulse insertion

LL_RTC_CALIB_INSERTPULSE_NONE

No RTCCLK pulses are added

LL_RTC_CALIB_INSERTPULSE_SET

One RTCCLK pulse is effectively inserted every 2exp11 pulses (frequency increased by 488.5 ppm)

Calibration low power

LL_RTC_CALIB_LOWPOWER_NONE

High conso mode

LL_RTC_CALIB_LOWPOWER_SET

low power mode

Calibration output

LL_RTC_CALIB_OUTPUT_NONE

Calibration output disabled

LL_RTC_CALIB_OUTPUT_1HZ

Calibration output is 1 Hz

LL_RTC_CALIB_OUTPUT_512HZ

Calibration output is 512 Hz

Calibration period

LL_RTC_CALIB_PERIOD_32SEC

Use a 32-second calibration cycle period

LL_RTC_CALIB_PERIOD_16SEC

Use a 16-second calibration cycle period

LL_RTC_CALIB_PERIOD_8SEC

Use a 8-second calibration cycle period

FORMAT

LL_RTC_FORMAT_BIN

Binary data format

LL_RTC_FORMAT_BCD

BCD data format

Get Flags Defines

LL_RTC_SCR_SSRUF

LL_RTC_SCR_ITSF

LL_RTC_SCR_TSOVF

LL_RTC_SCR_TSF

LL_RTC_SCR_WUTF

LL_RTC_SCR_ALRBF

LL_RTC_SCR_ALRAF

LL_RTC_ICSR_RECALPF

LL_RTC_ICSR_BCDU_2

LL_RTC_ICSR_BCDU_1

LL_RTC_ICSR_BCDU_0

LL_RTC_ICSR_BIN_1

LL_RTC_ICSR_BIN_0

LL_RTC_ICSR_INITF

LL_RTC_ICSR_RSF

LL_RTC_ICSR_INITS

LL_RTC_ICSR_SHPF

LL_RTC_ICSR_WUTWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR

24 hour/day format

LL_RTC_HOURFORMAT_AMPM

AM/PM hour format

INTERNAL TAMPER

LL_RTC_TAMPER_ITAMP1

Internal tamper 1: RTC supply voltage monitoring

LL_RTC_TAMPER_ITAMP2

Internal tamper 2: Temperature monitoring

LL_RTC_TAMPER_ITAMP3

Internal tamper 3: LSE monitoring

LL_RTC_TAMPER_ITAMP5

Internal tamper 5: RTC calendar overflow

LL_RTC_TAMPER_ITAMP6

Internal tamper 6: JTAG/SWD access when RDP > 0

LL_RTC_TAMPER_ITAMP7

Internal tamper 7: ADC4 analog watchdog monitoring 1

LL_RTC_TAMPER_ITAMP8

Internal tamper 8: Monotonic counter overflow

LL_RTC_TAMPER_ITAMP9

Internal tamper 9: Cryptographic IPs fault

LL_RTC_TAMPER_ITAMP11

Internal tamper 11: IWDG reset when tamper flag is set

LL_RTC_TAMPER_ITAMP12

Internal tamper 12: ADC4 analog watchdog monitoring 2

LL_RTC_TAMPER_ITAMP13

Internal tamper 13: ADC4 analog watchdog monitoring 3

IT Defines

LL_RTC_CR_TSIE

LL_RTC_CR_WUTIE

LL_RTC_CR_ALRBIE

LL_RTC_CR_ALRAIE

MONTH

LL_RTC_MONTH_JANUARY

January

LL_RTC_MONTH_FEBRUARY

February

LL_RTC_MONTH_MARCH
March

LL_RTC_MONTH_APRIL
April

LL_RTC_MONTH_MAY
May

LL_RTC_MONTH_JUNE
June

LL_RTC_MONTH_JULY
July

LL_RTC_MONTH_AUGUST
August

LL_RTC_MONTH_SEPTEMBER
September

LL_RTC_MONTH_OCTOBER
October

LL_RTC_MONTH_NOVEMBER
November

LL_RTC_MONTH_DECEMBER
December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH
Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW
Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

Privilege Backup register privilege zone

LL_RTC_PRIVILEGE_BKUP_ZONE_NONE

LL_RTC_PRIVILEGE_BKUP_ZONE_1

LL_RTC_PRIVILEGE_BKUP_ZONE_2

LL_RTC_PRIVILEGE_BKUP_ZONE_ALL

Privilege rtc features in case of LL_RTC_PRIVILEGE_FULL_NO.

LL_RTC_PRIVILEGE_FEATURE_INIT
Initialization feature is privilege

LL_RTC_PRIVILEGE_FEATURE_CAL
Calibration feature is privilege

LL_RTC_PRIVILEGE_FEATURE_TS
Time stamp feature is privilege

LL_RTC_PRIVILEGE_FEATURE_WUT
Wake up timer feature is privilege

LL_RTC_PRIVILEGE_FEATURE_ALRA
Alarm A feature is privilege

LL_RTC_PRIVILEGE_FEATURE_ALRB
Alarm B feature is privilege

Privilege full rtc

LL_RTC_PRIVILEGE_FULL_YES
RTC full privilege

LL_RTC_PRIVILEGE_FULL_NO
RTC is not full privilege, features can be unprivilege. See RTC_LL_EC_PRIVILEGE_RTC_FEATURE

Privilege full tamp

LL_TAMP_PRIVILEGE_FULL_YES
TAMP full privilege

LL_TAMP_PRIVILEGE_FULL_NO
TAMP is not privilege

Secure full rtc

LL_RTC_SECURE_FULL_YES
RTC full secure

LL_RTC_SECURE_FULL_NO
RTC is not full secure, features can be secure. See RTC_LL_EC_SECURE_RTC_FEATURE

Secure tamp

LL_TAMP_SECURE_FULL_YES
TAMP full secure

LL_TAMP_SECURE_FULL_NO
TAMP is not secure

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

TAMPER

LL_RTC_TAMPER_1
Tamper 1 input detection

LL_RTC_TAMPER_2
Tamper 2 input detection

LL_RTC_TAMPER_3
Tamper 3 input detection

LL_RTC_TAMPER_4

Tamper 4 input detection

LL_RTC_TAMPER_5

Tamper 5 input detection

LL_RTC_TAMPER_6

Tamper 6 input detection

LL_RTC_TAMPER_7

Tamper 7 input detection

LL_RTC_TAMPER_8

Tamper 8 input detection

TAMPER ACTIVE LEVEL

LL_RTC_TAMPER_ACTIVELEVEL_TAMP1

Tamper 1 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP2

Tamper 2 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP3

Tamper 3 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP4

Tamper 4 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP5

Tamper 5 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP6

Tamper 6 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP7

Tamper 7 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

LL_RTC_TAMPER_ACTIVELEVEL_TAMP8

Tamper 8 input falling edge (if TAMPFLT = 00) or staying high (if TAMPFLT != 00) triggers a tamper detection event

TAMPER DURATION

LL_RTC_TAMPER_DURATION_1RTCCLK

Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

LL_RTC_TAMPER_DURATION_2RTCCLK

Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

LL_RTC_TAMPER_DURATION_4RTCCLK

Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

LL_RTC_TAMPER_DURATION_8RTCCLK

Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

TAMPER FILTER

LL_RTC_TAMPER_FILTER_DISABLE

Tamper filter is disabled

LL_RTC_TAMPER_FILTER_2SAMPLE

Tamper is activated after 2 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_4SAMPLE

Tamper is activated after 4 consecutive samples at the active level

LL_RTC_TAMPER_FILTER_8SAMPLE

Tamper is activated after 8 consecutive samples at the active level.

TAMPER MASK

LL_RTC_TAMPER_MASK_TAMPER1

Tamper 1 event generates a trigger event. TAMP1F is masked and internally cleared by hardware.The backup registers are not erased

LL_RTC_TAMPER_MASK_TAMPER2

Tamper 2 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

LL_RTC_TAMPER_MASK_TAMPER3

Tamper 3 event generates a trigger event. TAMP2F is masked and internally cleared by hardware. The backup registers are not erased.

TAMPER NO ERASE

LL_RTC_TAMPER_NOERASE_TAMPER1

Tamper 1 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER2

Tamper 2 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER3

Tamper 3 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER4

Tamper 4 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER5

Tamper 5 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER6

Tamper 6 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER7

Tamper 7 event does not erase the backup registers.

LL_RTC_TAMPER_NOERASE_TAMPER8
Tamper 8 event does not erase the backup registers.

TAMPER SAMPLING FREQUENCY DIVIDER

LL_RTC_TAMPER_SAMPLFREQDIV_32768
Each of the tamper inputs are sampled with a frequency = RTCCLK / 32768

LL_RTC_TAMPER_SAMPLFREQDIV_16384
Each of the tamper inputs are sampled with a frequency = RTCCLK / 16384

LL_RTC_TAMPER_SAMPLFREQDIV_8192
Each of the tamper inputs are sampled with a frequency = RTCCLK / 8192

LL_RTC_TAMPER_SAMPLFREQDIV_4096
Each of the tamper inputs are sampled with a frequency = RTCCLK / 4096

LL_RTC_TAMPER_SAMPLFREQDIV_2048
Each of the tamper inputs are sampled with a frequency = RTCCLK / 2048

LL_RTC_TAMPER_SAMPLFREQDIV_1024
Each of the tamper inputs are sampled with a frequency = RTCCLK / 1024

LL_RTC_TAMPER_SAMPLFREQDIV_512
Each of the tamper inputs are sampled with a frequency = RTCCLK / 512

LL_RTC_TAMPER_SAMPLFREQDIV_256
Each of the tamper inputs are sampled with a frequency = RTCCLK / 256

TIMESTAMP EDGE

LL_RTC_TIMESTAMP_EDGE_RISING
RTC_TS input rising edge generates a time-stamp event

LL_RTC_TIMESTAMP_EDGE_FALLING
RTC_TS input falling edge generates a time-stamp even

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24
AM or 24-hour format

LL_RTC_TIME_FORMAT_PM
PM

TIMESTAMP TIME FORMAT

LL_RTC_TS_TIME_FORMAT_AM
AM or 24-hour format

LL_RTC_TS_TIME_FORMAT_PM
PM

UnSecure features rtc in case of LL_RTC_SECURE_FULL_NO.

LL_RTC_UNSECURE_FEATURE_INIT
Initialization feature is not secure

LL_RTC_UNSECURE_FEATURE_CAL
Calibration feature is not secure

LL_RTC_UNSECURE_FEATURE_TS
Time stamp feature is not secure

LL_RTC_UNSECURE_FEATURE_WUT
Wake up timer feature is not secure

LL_RTC_UNSECURE_FEATURE_ALRA
Alarm A feature is not secure

LL_RTC_UNSECURE_FEATURE_ALRB
Alarm B feature is not secure

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16
RTC/16 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_8
RTC/8 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_4
RTC/4 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_2
RTC/2 clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE
ck_spre (usually 1 Hz) clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE_WUT
ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY
Monday

LL_RTC_WEEKDAY_TUESDAY
Tuesday

LL_RTC_WEEKDAY_WEDNESDAY
Wednesday

LL_RTC_WEEKDAY_THURSDAY
Thrusday

LL_RTC_WEEKDAY_FRIDAY
Friday

LL_RTC_WEEKDAY_SATURDAY
Saturday

LL_RTC_WEEKDAY_SUNDAY
Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:

- Helper macro to retrieve Day in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:

- Helper macro to retrieve hour in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:

- Helper macro to retrieve minute in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:

- Helper macro to retrieve second in BCD format.

Parameters:

- __RTC_TIME__: RTC time returned by

Return value:

- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:

- Write a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

117 LL SPI Generic Driver

117.1 SPI Firmware driver registers structures

117.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32u5xx_ll_spi.h

Data Fields

- ***uint32_t TransferDirection***
- ***uint32_t Mode***
- ***uint32_t DataWidth***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***
- ***uint32_t NSS***
- ***uint32_t BaudRate***
- ***uint32_t BitOrder***
- ***uint32_t CRCCalculation***
- ***uint32_t CRCPoly***

Field Documentation

- ***uint32_t LL_SPI_InitTypeDef::TransferDirection***
  Specifies the SPI unidirectional or bidirectional data mode. This parameter can be a value of ***SPI_LL_EC_TRANSFER_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferDirection()**.
- ***uint32_t LL_SPI_InitTypeDef::Mode***
  Specifies the SPI mode (Master/Slave). This parameter can be a value of ***SPI_LL_EC_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetMode()**.
- ***uint32_t LL_SPI_InitTypeDef::DataWidth***
  Specifies the SPI data width. This parameter can be a value of ***SPI_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_SPI_SetDataWidth()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPolarity***
  Specifies the serial clock steady state. This parameter can be a value of ***SPI_LL_EC_POLARITY***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPolarity()**.
- ***uint32_t LL_SPI_InitTypeDef::ClockPhase***
  Specifies the clock active edge for the bit capture. This parameter can be a value of ***SPI_LL_EC_PHASE***.This feature can be modified afterwards using unitary function **LL_SPI_SetClockPhase()**.
- ***uint32_t LL_SPI_InitTypeDef::NSS***
  Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit.This parameter can be a value of ***SPI_LL_EC_NSS_MODE***.This feature can be modified afterwards using unitary function **LL_SPI_SetNSSMode()**.
- ***uint32_t LL_SPI_InitTypeDef::BaudRate***
  Specifies the BaudRate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of ***SPI_LL_EC_BAUDRATEPRESCALER***.
  Note:
  – The communication clock is derived from the master clock. The slave clock does not need to be set.

  This feature can be modified afterwards using unitary function **LL_SPI_SetBaudRatePrescaler()**.
- ***uint32_t LL_SPI_InitTypeDef::BitOrder***
  Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of ***SPI_LL_EC_BIT_ORDER***.This feature can be modified afterwards using unitary function **LL_SPI_SetTransferBitOrder()**.
- ***uint32_t LL_SPI_InitTypeDef::CRCCalculation***
  Specifies if the CRC calculation is enabled or not. This parameter can be a value of ***SPI_LL_EC_CRC_CALCULATION***.This feature can be modified afterwards using unitary functions **LL_SPI_EnableCRC()** and **LL_SPI_DisableCRC()**.

- ***uint32_t LL_SPI_InitTypeDef::CRCPoly***
  Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

117.2 SPI Firmware driver API description

The following section lists the various functions of the SPI library.

117.2.1 Detailed description of functions

LL_SPI_Enable

Function name

__STATIC_INLINE void LL_SPI_Enable (SPI_TypeDef * SPIx)

Function description

Enable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Enable

LL_SPI_Disable

Function name

__STATIC_INLINE void LL_SPI_Disable (SPI_TypeDef * SPIx)

Function description

Disable SPI peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- When disabling the SPI, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_Disable

LL_SPI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabled (const SPI_TypeDef * SPIx)

Function description

Check if SPI peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 SPE LL_SPI_IsEnabled

LL_SPI_EnableIOSwap

Function name

__STATIC_INLINE void LL_SPI_EnableIOSwap (SPI_TypeDef * SPIx)

Function description

Swap the MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_EnableIOSwap

LL_SPI_DisableIOSwap

Function name

__STATIC_INLINE void LL_SPI_DisableIOSwap (SPI_TypeDef * SPIx)

Function description

Restore default function for MOSI and MISO pin.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_DisableIOSwap

LL_SPI_IsEnabledIOSwap

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOSwap (const SPI_TypeDef * SPIx)

Function description

Check if MOSI and MISO pin are swapped.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 IOSWP LL_SPI_IsEnabledIOSwap

LL_SPI_EnableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_EnableGPIOControl (SPI_TypeDef * SPIx)

Function description

Enable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_EnableGPIOControl

LL_SPI_DisableGPIOControl

Function name

__STATIC_INLINE void LL_SPI_DisableGPIOControl (SPI_TypeDef * SPIx)

Function description

Disable GPIO control.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_DisableGPIOControl

LL_SPI_IsEnabledGPIOControl

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledGPIOControl (const SPI_TypeDef * SPIx)

Function description

Check if GPIO control is active.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 AFCNTR LL_SPI_IsEnabledGPIOControl

LL_SPI_SetMode

Function name

__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set SPI Mode to Master or Slave.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  – LL_SPI_MODE_MASTER
  – LL_SPI_MODE_SLAVE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_SetMode

LL_SPI_GetMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMode (const SPI_TypeDef * SPIx)

Function description

Get SPI Mode (Master or Slave)

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_MODE_MASTER
  – LL_SPI_MODE_SLAVE

Reference Manual to LL API cross reference:

- CFG2 MASTER LL_SPI_GetMode

LL_SPI_SetMasterSSIdleness

Function name

__STATIC_INLINE void LL_SPI_SetMasterSSIdleness (SPI_TypeDef * SPIx, uint32_t MasterSSIdleness)

Function description

Configure the Idleness applied by master between active edge of SS and first send data.

Parameters

- **SPIx:** SPI Instance
- **MasterSSIdleness:** This parameter can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_SetMasterSSIdleness

LL_SPI_GetMasterSSIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMasterSSIdleness (const SPI_TypeDef * SPIx)

Function description

Get the configured Idleness applied by master.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_SS_IDLENESS_00CYCLE
  - LL_SPI_SS_IDLENESS_01CYCLE
  - LL_SPI_SS_IDLENESS_02CYCLE
  - LL_SPI_SS_IDLENESS_03CYCLE
  - LL_SPI_SS_IDLENESS_04CYCLE
  - LL_SPI_SS_IDLENESS_05CYCLE
  - LL_SPI_SS_IDLENESS_06CYCLE
  - LL_SPI_SS_IDLENESS_07CYCLE
  - LL_SPI_SS_IDLENESS_08CYCLE
  - LL_SPI_SS_IDLENESS_09CYCLE
  - LL_SPI_SS_IDLENESS_10CYCLE
  - LL_SPI_SS_IDLENESS_11CYCLE
  - LL_SPI_SS_IDLENESS_12CYCLE
  - LL_SPI_SS_IDLENESS_13CYCLE
  - LL_SPI_SS_IDLENESS_14CYCLE
  - LL_SPI_SS_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MSSI LL_SPI_GetMasterSSIdleness

LL_SPI_SetInterDataIdleness

Function name

__STATIC_INLINE void LL_SPI_SetInterDataIdleness (SPI_TypeDef * SPIx, uint32_t MasterInterDataIdleness)

Function description

Configure the idleness applied by master between data frame.

Parameters

- **SPIx:** SPI Instance
- **MasterInterDataIdleness:** This parameter can be one of the following values:
  - LL_SPI_ID_IDLENESS_00CYCLE
  - LL_SPI_ID_IDLENESS_01CYCLE
  - LL_SPI_ID_IDLENESS_02CYCLE
  - LL_SPI_ID_IDLENESS_03CYCLE
  - LL_SPI_ID_IDLENESS_04CYCLE
  - LL_SPI_ID_IDLENESS_05CYCLE
  - LL_SPI_ID_IDLENESS_06CYCLE
  - LL_SPI_ID_IDLENESS_07CYCLE
  - LL_SPI_ID_IDLENESS_08CYCLE
  - LL_SPI_ID_IDLENESS_09CYCLE
  - LL_SPI_ID_IDLENESS_10CYCLE
  - LL_SPI_ID_IDLENESS_11CYCLE
  - LL_SPI_ID_IDLENESS_12CYCLE
  - LL_SPI_ID_IDLENESS_13CYCLE
  - LL_SPI_ID_IDLENESS_14CYCLE
  - LL_SPI_ID_IDLENESS_15CYCLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_GetInterDataIdleness

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInterDataIdleness (const SPI_TypeDef * SPIx)

Function description

Get the configured inter data idleness.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_ID_IDLENESS_00CYCLE
  – LL_SPI_ID_IDLENESS_01CYCLE
  – LL_SPI_ID_IDLENESS_02CYCLE
  – LL_SPI_ID_IDLENESS_03CYCLE
  – LL_SPI_ID_IDLENESS_04CYCLE
  – LL_SPI_ID_IDLENESS_05CYCLE
  – LL_SPI_ID_IDLENESS_06CYCLE
  – LL_SPI_ID_IDLENESS_07CYCLE
  – LL_SPI_ID_IDLENESS_08CYCLE
  – LL_SPI_ID_IDLENESS_09CYCLE
  – LL_SPI_ID_IDLENESS_10CYCLE
  – LL_SPI_ID_IDLENESS_11CYCLE
  – LL_SPI_ID_IDLENESS_12CYCLE
  – LL_SPI_ID_IDLENESS_13CYCLE
  – LL_SPI_ID_IDLENESS_14CYCLE
  – LL_SPI_ID_IDLENESS_15CYCLE

Reference Manual to LL API cross reference:

- CFG2 MIDI LL_SPI_SetInterDataIdleness

LL_SPI_SetTransferSize

Function name

__STATIC_INLINE void LL_SPI_SetTransferSize (SPI_TypeDef * SPIx, uint32_t Count)

Function description

Set transfer size.

Parameters

- **SPIx:** SPI Instance
- **Count:** 0..0xFFFF

Return values

- **None:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_SetTransferSize

LL_SPI_GetTransferSize

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferSize (const SPI_TypeDef * SPIx)

Function description

Get transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Notes

- Count is the number of frame to be transferred

Reference Manual to LL API cross reference:

- CR2 TSIZE LL_SPI_GetTransferSize

LL_SPI_EnableIOLock

Function name

__STATIC_INLINE void LL_SPI_EnableIOLock (SPI_TypeDef * SPIx)

Function description

Lock the AF configuration of associated IOs.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Once this bit is set, the AF configuration remains locked until a hardware reset occurs. the reset of the IOLock bit is done by hardware. for that, LL_SPI_DisableIOLock can not exist.

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_EnableIOLock

LL_SPI_IsEnabledIOLock

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIOLock (const SPI_TypeDef * SPIx)

Function description

Check if the AF configuration is locked.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 IOLOCK LL_SPI_IsEnabledIOLock

LL_SPI_SetTxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetTxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t TXCRCInitAll)

Function description

Set Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **TXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_SetTxCRCInitPattern

LL_SPI_GetTxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRCInitPattern (const SPI_TypeDef * SPIx)

Function description

Get Tx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 TCRCINI LL_SPI_GetTxCRCInitPattern

LL_SPI_SetRxCRCInitPattern

Function name

__STATIC_INLINE void LL_SPI_SetRxCRCInitPattern (SPI_TypeDef * SPIx, uint32_t RXCRCInitAll)

Function description

Set Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance
- **RXCRCInitAll:** This parameter can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_SetRxCRCInitPattern

LL_SPI_GetRxCRCInitPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRCInitPattern (const SPI_TypeDef * SPIx)

Function description

Get Rx CRC Initialization Pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN
  - LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

Reference Manual to LL API cross reference:

- CR1 RCRCINI LL_SPI_GetRxCRCInitPattern

LL_SPI_SetInternalSSLevel

Function name

__STATIC_INLINE void LL_SPI_SetInternalSSLevel (SPI_TypeDef * SPIx, uint32_t SSLevel)

Function description

Set internal SS input level ignoring what comes from PIN.

Parameters

- **SPIx:** SPI Instance
- **SSLevel:** This parameter can be one of the following values:
  - LL_SPI_SS_LEVEL_HIGH
  - LL_SPI_SS_LEVEL_LOW

Return values

- **None:**

Notes

- This configuration has effect only with config LL_SPI_NSS_SOFT

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_SetInternalSSLevel

LL_SPI_GetInternalSSLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetInternalSSLevel (const SPI_TypeDef * SPIx)

Function description

Get internal SS input level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_SS_LEVEL_HIGH
  – LL_SPI_SS_LEVEL_LOW

Reference Manual to LL API cross reference:

- CR1 SSI LL_SPI_GetInternalSSLevel

LL_SPI_EnableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_EnableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_EnableFullSizeCRC

LL_SPI_DisableFullSizeCRC

Function name

__STATIC_INLINE void LL_SPI_DisableFullSizeCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC computation on 33/17 bits.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_DisableFullSizeCRC

LL_SPI_IsEnabledFullSizeCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledFullSizeCRC (const SPI_TypeDef * SPIx)

Function description

Check if Enable CRC computation on 33/17 bits is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CRC33_17 LL_SPI_IsEnabledFullSizeCRC

LL_SPI_SuspendMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_SuspendMasterTransfer (SPI_TypeDef * SPIx)

Function description

Suspend an ongoing transfer for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSUSP LL_SPI_SuspendMasterTransfer

LL_SPI_StartMasterTransfer

Function name

__STATIC_INLINE void LL_SPI_StartMasterTransfer (SPI_TypeDef * SPIx)

Function description

Start effective transfer on wire for Master configuration.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_StartMasterTransfer

LL_SPI_IsActiveMasterTransfer

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveMasterTransfer (const SPI_TypeDef * SPIx)

Function description

Check if there is an unfinished master transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 CSTART LL_SPI_IsActiveMasterTransfer

LL_SPI_EnableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_EnableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Enable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_EnableMasterRxAutoSuspend

LL_SPI_DisableMasterRxAutoSuspend

Function name

__STATIC_INLINE void LL_SPI_DisableMasterRxAutoSuspend (SPI_TypeDef * SPIx)

Function description

Disable Master Rx auto suspend in case of overrun.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_DisableMasterRxAutoSuspend

LL_SPI_IsEnabledMasterRxAutoSuspend

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledMasterRxAutoSuspend (const SPI_TypeDef * SPIx)

Function description

Check if Master Rx auto suspend is activated.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CR1 MASRX LL_SPI_IsEnabledMasterRxAutoSuspend

LL_SPI_SetUDRConfiguration

Function name

__STATIC_INLINE void LL_SPI_SetUDRConfiguration (SPI_TypeDef * SPIx, uint32_t UDRConfig)

Function description

Set Underrun behavior.

Parameters

- **SPIx:** SPI Instance
- **UDRConfig:** This parameter can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_SetUDRConfiguration

LL_SPI_GetUDRConfiguration

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRConfiguration (const SPI_TypeDef * SPIx)

Function description

Get Underrun behavior.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_UDR_CONFIG_REGISTER_PATTERN
  - LL_SPI_UDR_CONFIG_LAST_RECEIVED

Reference Manual to LL API cross reference:

- CFG1 UDRCFG LL_SPI_GetUDRConfiguration

LL_SPI_SetStandard

Function name

__STATIC_INLINE void LL_SPI_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set Serial protocol used.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_SPI_PROTOCOL_MOTOROLA
  - LL_SPI_PROTOCOL_TI

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_SetStandard

LL_SPI_GetStandard

Function name

__STATIC_INLINE uint32_t LL_SPI_GetStandard (const SPI_TypeDef * SPIx)

Function description

Get Serial protocol used.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_PROTOCOL_MOTOROLA
  – LL_SPI_PROTOCOL_TI

Reference Manual to LL API cross reference:

- CFG2 SP LL_SPI_GetStandard

LL_SPI_SetClockPhase

Function name

__STATIC_INLINE void LL_SPI_SetClockPhase (SPI_TypeDef * SPIx, uint32_t ClockPhase)

Function description

Set Clock phase.

Parameters

- **SPIx:** SPI Instance
- **ClockPhase:** This parameter can be one of the following values:
  – LL_SPI_PHASE_1EDGE
  – LL_SPI_PHASE_2EDGE

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_SetClockPhase

LL_SPI_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPhase (const SPI_TypeDef * SPIx)

Function description

Get Clock phase.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_PHASE_1EDGE
  – LL_SPI_PHASE_2EDGE

Reference Manual to LL API cross reference:

- CFG2 CPHA LL_SPI_GetClockPhase

LL_SPI_SetClockPolarity

Function name

__STATIC_INLINE void LL_SPI_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set Clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  – LL_SPI_POLARITY_LOW
  – LL_SPI_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_SetClockPolarity

LL_SPI_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetClockPolarity (const SPI_TypeDef * SPIx)

Function description

Get Clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_POLARITY_LOW
  – LL_SPI_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetNSSPolarity

Function name

__STATIC_INLINE void LL_SPI_SetNSSPolarity (SPI_TypeDef * SPIx, uint32_t NSSPolarity)

Function description

Set NSS polarity.

Parameters

- **SPIx:** SPI Instance
- **NSSPolarity:** This parameter can be one of the following values:
  – LL_SPI_NSS_POLARITY_LOW
  – LL_SPI_NSS_POLARITY_HIGH

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_SetNSSPolarity

LL_SPI_GetNSSPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSPolarity (const SPI_TypeDef * SPIx)

Function description

Get NSS polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_NSS_POLARITY_LOW
  – LL_SPI_NSS_POLARITY_HIGH

Reference Manual to LL API cross reference:

- CFG2 SSIOP LL_SPI_GetNSSPolarity

LL_SPI_SetBaudRatePrescaler

Function name

__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t Baudrate)

Function description

Set Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance
- **Baudrate:** This parameter can be one of the following values:
  – LL_SPI_BAUDRATEPRESCALER_BYPASS
  – LL_SPI_BAUDRATEPRESCALER_DIV2
  – LL_SPI_BAUDRATEPRESCALER_DIV4
  – LL_SPI_BAUDRATEPRESCALER_DIV8
  – LL_SPI_BAUDRATEPRESCALER_DIV16
  – LL_SPI_BAUDRATEPRESCALER_DIV32
  – LL_SPI_BAUDRATEPRESCALER_DIV64
  – LL_SPI_BAUDRATEPRESCALER_DIV128
  – LL_SPI_BAUDRATEPRESCALER_DIV256

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. SPI BaudRate = fPCLK/Pescaler.

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_SetBaudRatePrescaler
- CFG1 BPASS LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (const SPI_TypeDef * SPIx)

Function description

Get Baudrate Prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_BAUDRATEPRESCALER_BYPASS
  – LL_SPI_BAUDRATEPRESCALER_DIV2
  – LL_SPI_BAUDRATEPRESCALER_DIV4
  – LL_SPI_BAUDRATEPRESCALER_DIV8
  – LL_SPI_BAUDRATEPRESCALER_DIV16
  – LL_SPI_BAUDRATEPRESCALER_DIV32
  – LL_SPI_BAUDRATEPRESCALER_DIV64
  – LL_SPI_BAUDRATEPRESCALER_DIV128
  – LL_SPI_BAUDRATEPRESCALER_DIV256

Reference Manual to LL API cross reference:

- CFG1 MBR LL_SPI_GetBaudRatePrescaler
- CFG1 BPASS LL_SPI_GetBaudRatePrescaler

LL_SPI_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_SPI_SetTransferBitOrder (SPI_TypeDef * SPIx, uint32_t BitOrder)

Function description

Set Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_SetTransferBitOrder

LL_SPI_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferBitOrder (const SPI_TypeDef * SPIx)

Function description

Get Transfer Bit Order.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_LSB_FIRST
  - LL_SPI_MSB_FIRST

Reference Manual to LL API cross reference:

- CFG2 LSBFRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name

__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)

Function description

Set Transfer Mode.

Parameters

- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_TX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled except for half duplex direction using LL_SPI_SetHalfDuplexDirection.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetTransferDirection
- CFG2 COMM LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (const SPI_TypeDef * SPIx)

Function description

Get Transfer Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_FULL_DUPLEX
  – LL_SPI_SIMPLEX_TX
  – LL_SPI_SIMPLEX_RX
  – LL_SPI_HALF_DUPLEX_RX
  – LL_SPI_HALF_DUPLEX_TX

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetTransferDirection
- CFG2 COMM LL_SPI_GetTransferDirection

LL_SPI_SetHalfDuplexDirection

Function name

__STATIC_INLINE void LL_SPI_SetHalfDuplexDirection (SPI_TypeDef * SPIx, uint32_t HalfDuplexDirection)

Function description

Set direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance
- **HalfDuplexDirection:** This parameter can be one of the following values:
  – LL_SPI_HALF_DUPLEX_RX
  – LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_SetHalfDuplexDirection

LL_SPI_GetHalfDuplexDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetHalfDuplexDirection (const SPI_TypeDef * SPIx)

Function description

Get direction for Half-Duplex Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Notes

- In master mode the MOSI pin is used and in slave mode the MISO pin is used for Half-Duplex.

Reference Manual to LL API cross reference:

- CR1 HDDIR LL_SPI_GetHalfDuplexDirection

LL_SPI_SetDataWidth

Function name

__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)

Function description

Set Frame Data Size.

Parameters

- **SPIx:** SPI Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (const SPI_TypeDef * SPIx)

Function description

Get Frame Data Size.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DATAWIDTH_4BIT
  - LL_SPI_DATAWIDTH_5BIT
  - LL_SPI_DATAWIDTH_6BIT
  - LL_SPI_DATAWIDTH_7BIT
  - LL_SPI_DATAWIDTH_8BIT
  - LL_SPI_DATAWIDTH_9BIT
  - LL_SPI_DATAWIDTH_10BIT
  - LL_SPI_DATAWIDTH_11BIT
  - LL_SPI_DATAWIDTH_12BIT
  - LL_SPI_DATAWIDTH_13BIT
  - LL_SPI_DATAWIDTH_14BIT
  - LL_SPI_DATAWIDTH_15BIT
  - LL_SPI_DATAWIDTH_16BIT
  - LL_SPI_DATAWIDTH_17BIT
  - LL_SPI_DATAWIDTH_18BIT
  - LL_SPI_DATAWIDTH_19BIT
  - LL_SPI_DATAWIDTH_20BIT
  - LL_SPI_DATAWIDTH_21BIT
  - LL_SPI_DATAWIDTH_22BIT
  - LL_SPI_DATAWIDTH_23BIT
  - LL_SPI_DATAWIDTH_24BIT
  - LL_SPI_DATAWIDTH_25BIT
  - LL_SPI_DATAWIDTH_26BIT
  - LL_SPI_DATAWIDTH_27BIT
  - LL_SPI_DATAWIDTH_28BIT
  - LL_SPI_DATAWIDTH_29BIT
  - LL_SPI_DATAWIDTH_30BIT
  - LL_SPI_DATAWIDTH_31BIT
  - LL_SPI_DATAWIDTH_32BIT

Reference Manual to LL API cross reference:

- CFG1 DSIZE LL_SPI_GetDataWidth

LL_SPI_SetFIFOThreshold

Function name

__STATIC_INLINE void LL_SPI_SetFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  – LL_SPI_FIFO_TH_01DATA
  – LL_SPI_FIFO_TH_02DATA
  – LL_SPI_FIFO_TH_03DATA
  – LL_SPI_FIFO_TH_04DATA
  – LL_SPI_FIFO_TH_05DATA
  – LL_SPI_FIFO_TH_06DATA
  – LL_SPI_FIFO_TH_07DATA
  – LL_SPI_FIFO_TH_08DATA
  – LL_SPI_FIFO_TH_09DATA
  – LL_SPI_FIFO_TH_10DATA
  – LL_SPI_FIFO_TH_11DATA
  – LL_SPI_FIFO_TH_12DATA
  – LL_SPI_FIFO_TH_13DATA
  – LL_SPI_FIFO_TH_14DATA
  – LL_SPI_FIFO_TH_15DATA
  – LL_SPI_FIFO_TH_16DATA

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_SetFIFOThreshold

LL_SPI_GetFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_SPI_GetFIFOThreshold (const SPI_TypeDef * SPIx)

Function description

Get threshold of FIFO that triggers a transfer event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FIFO_TH_01DATA
  - LL_SPI_FIFO_TH_02DATA
  - LL_SPI_FIFO_TH_03DATA
  - LL_SPI_FIFO_TH_04DATA
  - LL_SPI_FIFO_TH_05DATA
  - LL_SPI_FIFO_TH_06DATA
  - LL_SPI_FIFO_TH_07DATA
  - LL_SPI_FIFO_TH_08DATA
  - LL_SPI_FIFO_TH_09DATA
  - LL_SPI_FIFO_TH_10DATA
  - LL_SPI_FIFO_TH_11DATA
  - LL_SPI_FIFO_TH_12DATA
  - LL_SPI_FIFO_TH_13DATA
  - LL_SPI_FIFO_TH_14DATA
  - LL_SPI_FIFO_TH_15DATA
  - LL_SPI_FIFO_TH_16DATA

Reference Manual to LL API cross reference:

- CFG1 FTHLV LL_SPI_GetFIFOThreshold

LL_SPI_EnableCRC

Function name

__STATIC_INLINE void LL_SPI_EnableCRC (SPI_TypeDef * SPIx)

Function description

Enable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_EnableCRC

LL_SPI_DisableCRC

Function name

__STATIC_INLINE void LL_SPI_DisableCRC (SPI_TypeDef * SPIx)

Function description

Disable CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_DisableCRC

LL_SPI_IsEnabledCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledCRC (const SPI_TypeDef * SPIx)

Function description

Check if CRC is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 CRCEN LL_SPI_IsEnabledCRC

LL_SPI_SetCRCWidth

Function name

__STATIC_INLINE void LL_SPI_SetCRCWidth (SPI_TypeDef * SPIx, uint32_t CRCLength)

Function description

Set CRC Length.

Parameters

- **SPIx:** SPI Instance
- **CRCLength:** This parameter can be one of the following values:
  – LL_SPI_CRC_4BIT
  – LL_SPI_CRC_5BIT
  – LL_SPI_CRC_6BIT
  – LL_SPI_CRC_7BIT
  – LL_SPI_CRC_8BIT
  – LL_SPI_CRC_9BIT
  – LL_SPI_CRC_10BIT
  – LL_SPI_CRC_11BIT
  – LL_SPI_CRC_12BIT
  – LL_SPI_CRC_13BIT
  – LL_SPI_CRC_14BIT
  – LL_SPI_CRC_15BIT
  – LL_SPI_CRC_16BIT
  – LL_SPI_CRC_17BIT
  – LL_SPI_CRC_18BIT
  – LL_SPI_CRC_19BIT
  – LL_SPI_CRC_20BIT
  – LL_SPI_CRC_21BIT
  – LL_SPI_CRC_22BIT
  – LL_SPI_CRC_23BIT
  – LL_SPI_CRC_24BIT
  – LL_SPI_CRC_25BIT
  – LL_SPI_CRC_26BIT
  – LL_SPI_CRC_27BIT
  – LL_SPI_CRC_28BIT
  – LL_SPI_CRC_29BIT
  – LL_SPI_CRC_30BIT
  – LL_SPI_CRC_31BIT
  – LL_SPI_CRC_32BIT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled.

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_SetCRCWidth

LL_SPI_GetCRCWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCWidth (const SPI_TypeDef * SPIx)

Function description

Get CRC Length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_CRC_4BIT
  – LL_SPI_CRC_5BIT
  – LL_SPI_CRC_6BIT
  – LL_SPI_CRC_7BIT
  – LL_SPI_CRC_8BIT
  – LL_SPI_CRC_9BIT
  – LL_SPI_CRC_10BIT
  – LL_SPI_CRC_11BIT
  – LL_SPI_CRC_12BIT
  – LL_SPI_CRC_13BIT
  – LL_SPI_CRC_14BIT
  – LL_SPI_CRC_15BIT
  – LL_SPI_CRC_16BIT
  – LL_SPI_CRC_17BIT
  – LL_SPI_CRC_18BIT
  – LL_SPI_CRC_19BIT
  – LL_SPI_CRC_20BIT
  – LL_SPI_CRC_21BIT
  – LL_SPI_CRC_22BIT
  – LL_SPI_CRC_23BIT
  – LL_SPI_CRC_24BIT
  – LL_SPI_CRC_25BIT
  – LL_SPI_CRC_26BIT
  – LL_SPI_CRC_27BIT
  – LL_SPI_CRC_28BIT
  – LL_SPI_CRC_29BIT
  – LL_SPI_CRC_30BIT
  – LL_SPI_CRC_31BIT
  – LL_SPI_CRC_32BIT

Reference Manual to LL API cross reference:

- CFG1 CRCSIZE LL_SPI_GetCRCWidth

LL_SPI_SetNSSMode

Function name

__STATIC_INLINE void LL_SPI_SetNSSMode (SPI_TypeDef * SPIx, uint32_t NSS)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance
- **NSS:** This parameter can be one of the following values:
  – LL_SPI_NSS_SOFT
  – LL_SPI_NSS_HARD_INPUT
  – LL_SPI_NSS_HARD_OUTPUT

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_SetNSSMode
- CFG2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (const SPI_TypeDef * SPIx)

Function description

Set NSS Mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_NSS_SOFT
  – LL_SPI_NSS_HARD_INPUT
  – LL_SPI_NSS_HARD_OUTPUT

Reference Manual to LL API cross reference:

- CFG2 SSM LL_SPI_GetNSSMode
- CFG2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Enable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Disable NSS pulse mgt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This configuration can not be changed when SPI is enabled. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_DisableNSSPulseMgt

LL_SPI_IsEnabledNSSPulse

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledNSSPulse (const SPI_TypeDef * SPIx)

Function description

Check if NSS pulse is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG2 SSOM LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXP (const SPI_TypeDef * SPIx)

Function description

Check if there is enough data in FIFO to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXP LL_SPI_IsActiveFlag_RXP

LL_SPI_IsActiveFlag_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXP (const SPI_TypeDef * SPIx)

Function description

Check if there is enough space in FIFO to hold a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR TXP LL_SPI_IsActiveFlag_TXP

LL_SPI_IsActiveFlag_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_DXP (const SPI_TypeDef * SPIx)

Function description

Check if there enough space in FIFO to hold a full packet, AND enough data to read a full packet.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR DXP LL_SPI_IsActiveFlag_DXP

LL_SPI_IsActiveFlag_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_EOT (const SPI_TypeDef * SPIx)

Function description

Check that end of transfer event occurred.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR EOT LL_SPI_IsActiveFlag_EOT

LL_SPI_IsActiveFlag_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXTF (const SPI_TypeDef * SPIx)

Function description

Check that all required data has been filled in the fifo according to transfer size.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXTF LL_SPI_IsActiveFlag_TXTF

LL_SPI_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_UDR (const SPI_TypeDef * SPIx)

Function description

Get Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UDR LL_SPI_IsActiveFlag_UDR

LL_SPI_IsActiveFlag_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_CRCERR (const SPI_TypeDef * SPIx)

Function description

Get CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CRCE LL_SPI_IsActiveFlag_CRCERR

LL_SPI_IsActiveFlag_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_MODF (const SPI_TypeDef * SPIx)

Function description

Get Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_IsActiveFlag_MODF

LL_SPI_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_OVR (const SPI_TypeDef * SPIx)

Function description

Get Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_IsActiveFlag_OVR

LL_SPI_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get TI Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIFRE LL_SPI_IsActiveFlag_FRE

LL_SPI_IsActiveFlag_SUSP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_SUSP (const SPI_TypeDef * SPIx)

Function description

Check if a suspend operation is done.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR SUSP LL_SPI_IsActiveFlag_SUSP

LL_SPI_IsActiveFlag_TXC

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXC (const SPI_TypeDef * SPIx)

Function description

Check if last TxFIFO or CRC frame transmission is completed.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXC LL_SPI_IsActiveFlag_TXC

LL_SPI_IsActiveFlag_RXWNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXWNE (const SPI_TypeDef * SPIx)

Function description

Check if at least one 32-bit data is available in RxFIFO.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- SR RXWNE LL_SPI_IsActiveFlag_RXWNE

LL_SPI_GetRemainingDataFrames

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRemainingDataFrames (const SPI_TypeDef * SPIx)

Function description

Get number of data framed remaining in current TSIZE.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- SR CTSIZE LL_SPI_GetRemainingDataFrames

LL_SPI_GetRxFIFOPackingLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOPackingLevel (const SPI_TypeDef * SPIx)

Function description

Get RxFIFO packing Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_RX_FIFO_0PACKET
  – LL_SPI_RX_FIFO_1PACKET
  – LL_SPI_RX_FIFO_2PACKET
  – LL_SPI_RX_FIFO_3PACKET

Reference Manual to LL API cross reference:

- SR RXPLVL LL_SPI_GetRxFIFOPackingLevel

LL_SPI_ClearFlag_EOT

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_EOT (SPI_TypeDef * SPIx)

Function description

Clear End Of Transfer flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR EOTC LL_SPI_ClearFlag_EOT

LL_SPI_ClearFlag_TXTF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_TXTF (SPI_TypeDef * SPIx)

Function description

Clear TXTF flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TXTFC LL_SPI_ClearFlag_TXTF

LL_SPI_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_UDR (SPI_TypeDef * SPIx)

Function description

Clear Underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR UDRC LL_SPI_ClearFlag_UDR

LL_SPI_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_OVR (SPI_TypeDef * SPIx)

Function description

Clear Overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR OVRC LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_CRCERR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_CRCERR (SPI_TypeDef * SPIx)

Function description

Clear CRC error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CRCEC LL_SPI_ClearFlag_CRCERR

LL_SPI_ClearFlag_MODF

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_MODF (SPI_TypeDef * SPIx)

Function description

Clear Mode fault error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR MODFC LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_FRE (SPI_TypeDef * SPIx)

Function description

Clear Frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR TIFREC LL_SPI_ClearFlag_FRE

LL_SPI_ClearFlag_SUSP

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_SUSP (SPI_TypeDef * SPIx)

Function description

Clear SUSP flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR SUSPC LL_SPI_ClearFlag_SUSP

LL_SPI_EnableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_RXP (SPI_TypeDef * SPIx)

Function description

Enable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_EnableIT_RXP

LL_SPI_EnableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXP (SPI_TypeDef * SPIx)

Function description

Enable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_EnableIT_TXP

LL_SPI_EnableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_EnableIT_DXP (SPI_TypeDef * SPIx)

Function description

Enable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_EnableIT_DXP

LL_SPI_EnableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_EnableIT_EOT (SPI_TypeDef * SPIx)

Function description

Enable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_EnableIT_EOT

LL_SPI_EnableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Enable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_EnableIT_TXTF

LL_SPI_EnableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_UDR (SPI_TypeDef * SPIx)

Function description

Enable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_EnableIT_UDR

LL_SPI_EnableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_OVR (SPI_TypeDef * SPIx)

Function description

Enable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_EnableIT_OVR

LL_SPI_EnableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Enable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_EnableIT_CRCERR

LL_SPI_EnableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_FRE (SPI_TypeDef * SPIx)

Function description

Enable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_EnableIT_FRE

LL_SPI_EnableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_EnableIT_MODF (SPI_TypeDef * SPIx)

Function description

Enable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_EnableIT_MODF

LL_SPI_DisableIT_RXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_RXP (SPI_TypeDef * SPIx)

Function description

Disable Rx Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_DisableIT_RXP

LL_SPI_DisableIT_TXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXP (SPI_TypeDef * SPIx)

Function description

Disable Tx Packet space available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_DisableIT_TXP

LL_SPI_DisableIT_DXP

Function name

__STATIC_INLINE void LL_SPI_DisableIT_DXP (SPI_TypeDef * SPIx)

Function description

Disable Duplex Packet available IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_DisableIT_DXP

LL_SPI_DisableIT_EOT

Function name

__STATIC_INLINE void LL_SPI_DisableIT_EOT (SPI_TypeDef * SPIx)

Function description

Disable End Of Transfer IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_DisableIT_EOT

LL_SPI_DisableIT_TXTF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXTF (SPI_TypeDef * SPIx)

Function description

Disable TXTF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_DisableIT_TXTF

LL_SPI_DisableIT_UDR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_UDR (SPI_TypeDef * SPIx)

Function description

Disable Underrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_DisableIT_UDR

LL_SPI_DisableIT_OVR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_OVR (SPI_TypeDef * SPIx)

Function description

Disable Overrun IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_DisableIT_OVR

LL_SPI_DisableIT_CRCERR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_CRCERR (SPI_TypeDef * SPIx)

Function description

Disable CRC Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_DisableIT_CRCERR

LL_SPI_DisableIT_FRE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_FRE (SPI_TypeDef * SPIx)

Function description

Disable TI Frame Format Error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_DisableIT_FRE

LL_SPI_DisableIT_MODF

Function name

__STATIC_INLINE void LL_SPI_DisableIT_MODF (SPI_TypeDef * SPIx)

Function description

Disable MODF IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_DisableIT_MODF

LL_SPI_IsEnabledIT_RXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXP (const SPI_TypeDef * SPIx)

Function description

Check if Rx Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER RXPIE LL_SPI_IsEnabledIT_RXP

LL_SPI_IsEnabledIT_TXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXP (const SPI_TypeDef * SPIx)

Function description

Check if Tx Packet space available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXPIE LL_SPI_IsEnabledIT_TXP

LL_SPI_IsEnabledIT_DXP

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_DXP (const SPI_TypeDef * SPIx)

Function description

Check if Duplex Packet available IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER DXPIE LL_SPI_IsEnabledIT_DXP

LL_SPI_IsEnabledIT_EOT

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_EOT (const SPI_TypeDef * SPIx)

Function description

Check if End Of Transfer IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER EOTIE LL_SPI_IsEnabledIT_EOT

LL_SPI_IsEnabledIT_TXTF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXTF (const SPI_TypeDef * SPIx)

Function description

Check if TXTF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TXTFIE LL_SPI_IsEnabledIT_TXTF

LL_SPI_IsEnabledIT_UDR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_UDR (const SPI_TypeDef * SPIx)

Function description

Check if Underrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER UDRIE LL_SPI_IsEnabledIT_UDR

LL_SPI_IsEnabledIT_OVR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_OVR (const SPI_TypeDef * SPIx)

Function description

Check if Overrun IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER OVRIE LL_SPI_IsEnabledIT_OVR

LL_SPI_IsEnabledIT_CRCERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_CRCERR (const SPI_TypeDef * SPIx)

Function description

Check if CRC Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER CRCEIE LL_SPI_IsEnabledIT_CRCERR

LL_SPI_IsEnabledIT_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_FRE (const SPI_TypeDef * SPIx)

Function description

Check if TI Frame Format Error IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER TIFREIE LL_SPI_IsEnabledIT_FRE

LL_SPI_IsEnabledIT_MODF

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_MODF (const SPI_TypeDef * SPIx)

Function description

Check if MODF IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- IER MODFIE LL_SPI_IsEnabledIT_MODF

LL_SPI_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_EnableDMAReq_RX

LL_SPI_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_DisableDMAReq_RX

LL_SPI_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_RX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Rx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_SPI_IsEnabledDMAReq_RX

LL_SPI_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_EnableDMAReq_TX

LL_SPI_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_SPI_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_DisableDMAReq_TX

LL_SPI_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledDMAReq_TX (const SPI_TypeDef * SPIx)

Function description

Check if DMA Tx is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0)

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_DMA_GetTxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetTxRegAddr (const SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- TXDR TXDR LL_SPI_DMA_GetTxRegAddr

LL_SPI_DMA_GetRxRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRxRegAddr (const SPI_TypeDef * SPIx)

Function description

Get the data register address used for DMA transfer.

Parameters

- **SPIx:** SPI Instance

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RXDR RXDR LL_SPI_DMA_GetRxRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData16

LL_SPI_ReceiveData32

Function name

__STATIC_INLINE uint32_t LL_SPI_ReceiveData32 (SPI_TypeDef * SPIx)

Function description

Read Data Register.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXDR . LL_SPI_ReceiveData32

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData16

LL_SPI_TransmitData32

Function name

__STATIC_INLINE void LL_SPI_TransmitData32 (SPI_TypeDef * SPIx, uint32_t TxData)

Function description

Write Data Register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TXDR . LL_SPI_TransmitData32

LL_SPI_SetCRCPolynomial

Function name

__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)

Function description

Set polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance
- **CRCPoly:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_SetCRCPolynomial

LL_SPI_GetCRCPolynomial

Function name

__STATIC_INLINE uint32_t LL_SPI_GetCRCPolynomial (const SPI_TypeDef * SPIx)

Function description

Get polynomial for CRC calcul.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- CRCPOLY CRCPOLY LL_SPI_GetCRCPolynomial

LL_SPI_SetUDRPattern

Function name

__STATIC_INLINE void LL_SPI_SetUDRPattern (SPI_TypeDef * SPIx, uint32_t Pattern)

Function description

Set the underrun pattern.

Parameters

- **SPIx:** SPI Instance
- **Pattern:** 0..0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_SetUDRPattern

LL_SPI_GetUDRPattern

Function name

__STATIC_INLINE uint32_t LL_SPI_GetUDRPattern (const SPI_TypeDef * SPIx)

Function description

Get the underrun pattern.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- UDRDR UDRDR LL_SPI_GetUDRPattern

LL_SPI_GetRxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxCRC (const SPI_TypeDef * SPIx)

Function description

Get Rx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- RXCRCR RXCRC LL_SPI_GetRxCRC

LL_SPI_GetTxCRC

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxCRC (const SPI_TypeDef * SPIx)

Function description

Get Tx CRC.

Parameters

- **SPIx:** SPI Instance

Return values

- **0..0xFFFFFFFF:**

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

LL_SPI_Enable_SelectedTrigger

Function name

__STATIC_INLINE void LL_SPI_Enable_SelectedTrigger (SPI_TypeDef * SPIx)

Function description

Enable Selected Trigger.

Parameters

- **SPIx:** SPI Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_SPI_Enable_SelectedTrigger

LL_SPI_Disable_SelectedTrigger

Function name

__STATIC_INLINE void LL_SPI_Disable_SelectedTrigger (SPI_TypeDef * SPIx)

Function description

Disable Selected Trigger.

Parameters

- **SPIx:** SPI Instance.

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_SPI_Disable_SelectedTrigger

LL_SPI_IsEnabled_SelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabled_SelectedTrigger (const SPI_TypeDef * SPIx)

Function description

Indicate if selected Trigger is disabled or enabled.

Parameters

- **SPIx:** SPI Instance.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_SPI_IsEnabled_SelectedTrigger

LL_SPI_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_SPI_SetTriggerPolarity (SPI_TypeDef * SPIx, uint32_t Polarity)

Function description

Set the trigger polarity.

Parameters

- **SPIx:** SPI Instance.
- **Polarity:** This parameter can be one of the following values:
  – LL_SPI_TRIG_POLARITY_RISING
  – LL_SPI_TRIG_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_SPI_SetTriggerPolarity

LL_SPI_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTriggerPolarity (const SPI_TypeDef * SPIx)

Function description

Get the trigger polarity.

Parameters

- **SPIx:** SPI Instance.

Return values

- **Returned:** value can be one of the following values:
  – LL_SPI_TRIG_POLARITY_RISING
  – LL_SPI_TRIG_POLARITY_FALLING

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_SPI_GetTriggerPolarity

LL_SPI_SetSelectedTrigger

Function name

__STATIC_INLINE void LL_SPI_SetSelectedTrigger (SPI_TypeDef * SPIx, uint32_t Trigger)

Function description

Set the selected trigger.

Parameters

- **SPIx:** SPI Instance.
- **Trigger:** This parameter can be one of the following values:
  – LL_SPI_GRP1_GPDMA_CH0_TCF_TRG
  – LL_SPI_GRP1_GPDMA_CH1_TCF_TRG
  – LL_SPI_GRP1_GPDMA_CH2_TCF_TRG
  – LL_SPI_GRP1_GPDMA_CH3_TCF_TRG
  – LL_SPI_GRP1_EXTI4_TRG
  – LL_SPI_GRP1_EXTI9_TRG
  – LL_SPI_GRP1_LPTIM1_CH1_TRG
  – LL_SPI_GRP1_LPTIM2_CH1_TRG
  – LL_SPI_GRP1_COMP1_TRG
  – LL_SPI_GRP1_COMP2_TRG
  – LL_SPI_GRP1_RTC_ALRA_TRG
  – LL_SPI_GRP1_RTC_WUT_TRG
  – LL_SPI_GRP2_LPDMA_CH0_TCF_TRG
  – LL_SPI_GRP2_LPDMA_CH1_TCF_TRG
  – LL_SPI_GRP2_LPDMA_CH2_TCF_TRG
  – LL_SPI_GRP2_LPDMA_CH3_TCF_TRG
  – LL_SPI_GRP2_EXTI4_TRG
  – LL_SPI_GRP2_EXTI8_TRG
  – LL_SPI_GRP2_LPTIM1_CH1_TRG
  – LL_SPI_GRP2_LPTIM3_CH1_TRG
  – LL_SPI_GRP2_COMP1_TRG
  – LL_SPI_GRP2_COMP2_TRG
  – LL_SPI_GRP2_RTC_ALRA_TRG
  – LL_SPI_GRP2_RTC_WUT_TRG

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_SPI_SetSelectedTrigger

LL_SPI_GetSelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_SPI_GetSelectedTrigger (const SPI_TypeDef * SPIx)

Function description

Get the selected trigger.

Parameters

- **SPIx:** SPI Instance.

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_GRP1_GPDMA_CH0_TCF_TRG
  - LL_SPI_GRP1_GPDMA_CH1_TCF_TRG
  - LL_SPI_GRP1_GPDMA_CH2_TCF_TRG
  - LL_SPI_GRP1_GPDMA_CH3_TCF_TRG
  - LL_SPI_GRP1_EXTI4_TRG
  - LL_SPI_GRP1_EXTI9_TRG
  - LL_SPI_GRP1_LPTIM1_CH1_TRG
  - LL_SPI_GRP1_LPTIM2_CH1_TRG
  - LL_SPI_GRP1_COMP1_TRG
  - LL_SPI_GRP1_COMP2_TRG
  - LL_SPI_GRP1_RTC_ALRA_TRG
  - LL_SPI_GRP1_RTC_WUT_TRG
  - LL_SPI_GRP2_LPDMA_CH0_TCF_TRG
  - LL_SPI_GRP2_LPDMA_CH1_TCF_TRG
  - LL_SPI_GRP2_LPDMA_CH2_TCF_TRG
  - LL_SPI_GRP2_LPDMA_CH3_TCF_TRG
  - LL_SPI_GRP2_EXTI4_TRG
  - LL_SPI_GRP2_EXTI8_TRG
  - LL_SPI_GRP2_LPTIM1_CH1_TRG
  - LL_SPI_GRP2_LPTIM3_CH1_TRG
  - LL_SPI_GRP2_COMP1_TRG
  - LL_SPI_GRP2_COMP2_TRG
  - LL_SPI_GRP2_RTC_ALRA_TRG
  - LL_SPI_GRP2_RTC_WUT_TRG

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_SPI_GetSelectedTrigger

LL_SPI_DeInit

Function name

ErrorStatus LL_SPI_DeInit (const SPI_TypeDef * SPIx)

Function description

De-initialize the SPI registers to their default reset values.

Parameters

- **SPIx:** SPI Instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: SPI registers are de-initialized
  – ERROR: SPI registers are not de-initialized

LL_SPI_Init

Function name

ErrorStatus LL_SPI_Init (SPI_TypeDef * SPIx, LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Initialize the SPI registers according to the specified parameters in SPI_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI IP should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_SPI_StructInit

Function name

void LL_SPI_StructInit (LL_SPI_InitTypeDef * SPI_InitStruct)

Function description

Set each LL_SPI_InitTypeDef field to default value.

Parameters

- **SPI_InitStruct:** pointer to a LL_SPI_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

117.3 SPI Firmware driver defines

The following section lists the various define and macros of the module.

117.3.1 SPI

SPI

Autonomous Trigger Polarity

LL_SPI_TRIG_POLARITY_RISING

SPI triggered on rising edge

LL_SPI_TRIG_POLARITY_FALLING

SPI triggered on falling edge

Autonomous Trigger selection

LL_SPI_TRIG_GRP1

Trigger Group for SPI1 and SPI2

LL_SPI_TRIG_GRP2
Trigger Group for SPI3

LL_SPI_GRP1_GPDMA_CH0_TCF_TRG
< HW Trigger signal is GPDMA_CH0_TRG HW Trigger signal is GPDMA_CH1_TRG

LL_SPI_GRP1_GPDMA_CH1_TCF_TRG
HW Trigger signal is GPDMA_CH2_TRG

LL_SPI_GRP1_GPDMA_CH2_TCF_TRG
HW Trigger signal is GPDMA_CH3_TRG

LL_SPI_GRP1_GPDMA_CH3_TCF_TRG
HW Trigger signal is EXTI4_TRG

LL_SPI_GRP1_EXTI4_TRG
HW Trigger signal is EXTI9_TRG

LL_SPI_GRP1_EXTI9_TRG
HW Trigger signal is LPTIM1_CH1_TRG

LL_SPI_GRP1_LPTIM1_CH1_TRG
HW Trigger signal is LPTIM2_CH1_TRG

LL_SPI_GRP1_LPTIM2_CH1_TRG
HW Trigger signal is COMP1_TRG

LL_SPI_GRP1_COMP1_TRG
HW Trigger signal is COMP2_TRG

LL_SPI_GRP1_COMP2_TRG
HW Trigger signal is RTC_ALRA_TRG

LL_SPI_GRP1_RTC_ALRA_TRG
HW Trigger signal is RTC_WUT_TRG

LL_SPI_GRP1_RTC_WUT_TRG
HW Trigger signal is LPDMA_CH0_TRG

LL_SPI_GRP2_LPDMA_CH0_TCF_TRG
HW Trigger signal is LPDMA_CH1_TRG

LL_SPI_GRP2_LPDMA_CH1_TCF_TRG
HW Trigger signal is LPDMA_CH2_TRG

LL_SPI_GRP2_LPDMA_CH2_TCF_TRG
HW Trigger signal is LPDMA_CH3_TRG

LL_SPI_GRP2_LPDMA_CH3_TCF_TRG
HW Trigger signal is EXTI4_TRG

LL_SPI_GRP2_EXTI4_TRG
HW Trigger signal is EXTI8_TRG

LL_SPI_GRP2_EXTI8_TRG
HW Trigger signal is LPTIM1_CH1_TRG

LL_SPI_GRP2_LPTIM1_CH1_TRG

HW Trigger signal is LPTIM3_CH1_TRG

LL_SPI_GRP2_LPTIM3_CH1_TRG

HW Trigger signal is COMP1_TRG

LL_SPI_GRP2_COMP1_TRG

HW Trigger signal is COMP2_TRG

LL_SPI_GRP2_COMP2_TRG

HW Trigger signal is RTC_ALRA_TRG

LL_SPI_GRP2_RTC_ALRA_TRG

HW Trigger signal is RTC_WUT_TRG

LL_SPI_GRP2_RTC_WUT_TRG

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_BYPASS

LL_SPI_BAUDRATEPRESCALER_DIV2

LL_SPI_BAUDRATEPRESCALER_DIV4

LL_SPI_BAUDRATEPRESCALER_DIV8

LL_SPI_BAUDRATEPRESCALER_DIV16

LL_SPI_BAUDRATEPRESCALER_DIV32

LL_SPI_BAUDRATEPRESCALER_DIV64

LL_SPI_BAUDRATEPRESCALER_DIV128

LL_SPI_BAUDRATEPRESCALER_DIV256

Bit Order

LL_SPI_LSB_FIRST

LL_SPI_MSB_FIRST

CRC

LL_SPI_CRC_4BIT

LL_SPI_CRC_5BIT

LL_SPI_CRC_6BIT

LL_SPI_CRC_7BIT

LL_SPI_CRC_8BIT

LL_SPI_CRC_9BIT

LL_SPI_CRC_10BIT

LL_SPI_CRC_11BIT

LL_SPI_CRC_12BIT

LL_SPI_CRC_13BIT

LL_SPI_CRC_14BIT

LL_SPI_CRC_15BIT

LL_SPI_CRC_16BIT

LL_SPI_CRC_17BIT

LL_SPI_CRC_18BIT

LL_SPI_CRC_19BIT

LL_SPI_CRC_20BIT

LL_SPI_CRC_21BIT

LL_SPI_CRC_22BIT

LL_SPI_CRC_23BIT

LL_SPI_CRC_24BIT

LL_SPI_CRC_25BIT

LL_SPI_CRC_26BIT

LL_SPI_CRC_27BIT

LL_SPI_CRC_28BIT

LL_SPI_CRC_29BIT

LL_SPI_CRC_30BIT

LL_SPI_CRC_31BIT

LL_SPI_CRC_32BIT

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE

CRC calculation disabled

LL_SPI_CRCCALCULATION_ENABLE

CRC calculation enabled

Data Width

LL_SPI_DATAWIDTH_4BIT

LL_SPI_DATAWIDTH_5BIT

LL_SPI_DATAWIDTH_6BIT

LL_SPI_DATAWIDTH_7BIT

LL_SPI_DATAWIDTH_8BIT

LL_SPI_DATAWIDTH_9BIT

LL_SPI_DATAWIDTH_10BIT

LL_SPI_DATAWIDTH_11BIT

LL_SPI_DATAWIDTH_12BIT

LL_SPI_DATAWIDTH_13BIT

LL_SPI_DATAWIDTH_14BIT

LL_SPI_DATAWIDTH_15BIT

LL_SPI_DATAWIDTH_16BIT

LL_SPI_DATAWIDTH_17BIT

LL_SPI_DATAWIDTH_18BIT

LL_SPI_DATAWIDTH_19BIT

LL_SPI_DATAWIDTH_20BIT

LL_SPI_DATAWIDTH_21BIT

LL_SPI_DATAWIDTH_22BIT

LL_SPI_DATAWIDTH_23BIT

LL_SPI_DATAWIDTH_24BIT

LL_SPI_DATAWIDTH_25BIT

LL_SPI_DATAWIDTH_26BIT

LL_SPI_DATAWIDTH_27BIT

LL_SPI_DATAWIDTH_28BIT

LL_SPI_DATAWIDTH_29BIT

LL_SPI_DATAWIDTH_30BIT

LL_SPI_DATAWIDTH_31BIT

LL_SPI_DATAWIDTH_32BIT

FIFO Threshold

LL_SPI_FIFO_TH_01DATA

LL_SPI_FIFO_TH_02DATA

LL_SPI_FIFO_TH_03DATA

LL_SPI_FIFO_TH_04DATA

LL_SPI_FIFO_TH_05DATA

LL_SPI_FIFO_TH_06DATA

LL_SPI_FIFO_TH_07DATA

LL_SPI_FIFO_TH_08DATA

LL_SPI_FIFO_TH_09DATA

LL_SPI_FIFO_TH_10DATA

LL_SPI_FIFO_TH_11DATA

LL_SPI_FIFO_TH_12DATA

LL_SPI_FIFO_TH_13DATA

LL_SPI_FIFO_TH_14DATA

LL_SPI_FIFO_TH_15DATA

LL_SPI_FIFO_TH_16DATA

Get Flags Defines

LL_SPI_SR_RXP

LL_SPI_SR_TXP

LL_SPI_SR_DXP

LL_SPI_SR_EOT

LL_SPI_SR_TXTF

LL_SPI_SR_UDR

LL_SPI_SR_CRCE

LL_SPI_SR_MODF

LL_SPI_SR_OVR

LL_SPI_SR_TIFRE

LL_SPI_SR_SUSP

LL_SPI_SR_TXC

LL_SPI_SR_RXWNE

Master Inter-Data Idleness

LL_SPI_ID_IDLENESS_00CYCLE

LL_SPI_ID_IDLENESS_01CYCLE

LL_SPI_ID_IDLENESS_02CYCLE

LL_SPI_ID_IDLENESS_03CYCLE

LL_SPI_ID_IDLENESS_04CYCLE

LL_SPI_ID_IDLENESS_05CYCLE

LL_SPI_ID_IDLENESS_06CYCLE

LL_SPI_ID_IDLENESS_07CYCLE

LL_SPI_ID_IDLENESS_08CYCLE

LL_SPI_ID_IDLENESS_09CYCLE

LL_SPI_ID_IDLENESS_10CYCLE

LL_SPI_ID_IDLENESS_11CYCLE

LL_SPI_ID_IDLENESS_12CYCLE

LL_SPI_ID_IDLENESS_13CYCLE

LL_SPI_ID_IDLENESS_14CYCLE

LL_SPI_ID_IDLENESS_15CYCLE

IT Defines

LL_SPI_IER_RXPIE

LL_SPI_IER_TXPIE

LL_SPI_IER_DXPIE

LL_SPI_IER_EOTIE

LL_SPI_IER_TXTFIE

LL_SPI_IER_UDRIE

LL_SPI_IER_OVRIE

LL_SPI_IER_CRCEIE

LL_SPI_IER_TIFREIE

LL_SPI_IER_MODFIE

Mode

LL_SPI_MODE_MASTER

LL_SPI_MODE_SLAVE

NSS Mode

LL_SPI_NSS_SOFT

LL_SPI_NSS_HARD_INPUT

LL_SPI_NSS_HARD_OUTPUT

NSS Polarity

LL_SPI_NSS_POLARITY_LOW

LL_SPI_NSS_POLARITY_HIGH

Phase

LL_SPI_PHASE_1EDGE

LL_SPI_PHASE_2EDGE

Polarity

LL_SPI_POLARITY_LOW

LL_SPI_POLARITY_HIGH

Protocol

LL_SPI_PROTOCOL_MOTOROLA

LL_SPI_PROTOCOL_TI

RXCRC Init All

LL_SPI_RXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_RXCRCINIT_ALL_ONES_PATTERN

RxFIFO Packing LeVel

LL_SPI_RX_FIFO_0PACKET

LL_SPI_RX_FIFO_1PACKET

LL_SPI_RX_FIFO_2PACKET

LL_SPI_RX_FIFO_3PACKET

SS Idleness

LL_SPI_SS_IDLENESS_00CYCLE

LL_SPI_SS_IDLENESS_01CYCLE

LL_SPI_SS_IDLENESS_02CYCLE

LL_SPI_SS_IDLENESS_03CYCLE

LL_SPI_SS_IDLENESS_04CYCLE

LL_SPI_SS_IDLENESS_05CYCLE

LL_SPI_SS_IDLENESS_06CYCLE

LL_SPI_SS_IDLENESS_07CYCLE

LL_SPI_SS_IDLENESS_08CYCLE

LL_SPI_SS_IDLENESS_09CYCLE

LL_SPI_SS_IDLENESS_10CYCLE

LL_SPI_SS_IDLENESS_11CYCLE

LL_SPI_SS_IDLENESS_12CYCLE

LL_SPI_SS_IDLENESS_13CYCLE

LL_SPI_SS_IDLENESS_14CYCLE

LL_SPI_SS_IDLENESS_15CYCLE

SS Level

LL_SPI_SS_LEVEL_HIGH

LL_SPI_SS_LEVEL_LOW

Transfer Mode

LL_SPI_FULL_DUPLEX

LL_SPI_SIMPLEX_TX

LL_SPI_SIMPLEX_RX

LL_SPI_HALF_DUPLEX_RX

LL_SPI_HALF_DUPLEX_TX

TXCRC Init All

LL_SPI_TXCRCINIT_ALL_ZERO_PATTERN

LL_SPI_TXCRCINIT_ALL_ONES_PATTERN

UDR Config Register

LL_SPI_UDR_CONFIG_REGISTER_PATTERN

LL_SPI_UDR_CONFIG_LAST_RECEIVED

Common Write and read registers Macros

LL_SPI_WriteReg

Description:
- Write a value in SPI register.

Parameters:
- __INSTANCE__: SPI Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_SPI_ReadReg

Description:
- Read a value in SPI register.

Parameters:
- __INSTANCE__: SPI Instance
- __REG__: Register to be read

Return value:
- Register: value

118 LL SYSTEM Generic Driver

118.1 SYSTEM Firmware driver API description

The following section lists the various functions of the SYSTEM library.

118.1.1 Detailed description of functions

LL_SYSCFG_EnableAnalogSwitchVdd

Function name

__STATIC_INLINE void LL_SYSCFG_EnableAnalogSwitchVdd (void)

Function description

Enable I/O analog switches supplied by VDD.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 ANASWVDD LL_SYSCFG_EnableAnalogSwitchVdd

LL_SYSCFG_DisableAnalogSwitchVdd

Function name

__STATIC_INLINE void LL_SYSCFG_DisableAnalogSwitchVdd (void)

Function description

Disable I/O analog switches supplied by VDD.

Return values

- **None:**

Notes

- I/O analog switches are supplied by VDDA or booster when booster in on. Dedicated voltage booster (supplied by VDD) is the recommended configuration with low VDDA voltage operation.

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 ANASWVDD LL_SYSCFG_DisableAnalogSwitchVdd

LL_SYSCFG_EnableAnalogBooster

Function name

__STATIC_INLINE void LL_SYSCFG_EnableAnalogBooster (void)

Function description

Enable I/O analog switch voltage booster.

Return values

- **None:**

Notes

- When voltage booster is enabled, I/O analog switches are supplied by a dedicated voltage booster, from VDD power domain. This is the recommended configuration with low VDDA voltage operation.
- The I/O analog switch voltage booster is relevant for peripherals using I/O in analog input: ADC, COMP, OPAMP. However, COMP and OPAMP inputs have a high impedance and voltage booster do not impact performance significantly. Therefore, the voltage booster is mainly intended for usage with ADC.

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 BOOSTEN LL_SYSCFG_EnableAnalogBooster

LL_SYSCFG_DisableAnalogBooster

Function name

__STATIC_INLINE void LL_SYSCFG_DisableAnalogBooster (void)

Function description

Disable I/O analog switch voltage booster.

Return values

- **None:**

Notes

- When voltage booster is enabled, I/O analog switches are supplied by a dedicated voltage booster, from VDD power domain. This is the recommended configuration with low VDDA voltage operation.
- The I/O analog switch voltage booster is relevant for peripherals using I/O in analog input: ADC, COMP, OPAMP. However, COMP and OPAMP inputs have a high impedance and voltage booster do not impact performance significantly. Therefore, the voltage booster is mainly intended for usage with ADC.

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 BOOSTEN LL_SYSCFG_DisableAnalogBooster

LL_SYSCFG_EnableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_FASTMODEPLUS_PB6
  - LL_SYSCFG_FASTMODEPLUS_PB7
  - LL_SYSCFG_FASTMODEPLUS_PB8
  - LL_SYSCFG_FASTMODEPLUS_PB9

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 PBx_FMP LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_DisableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_FASTMODEPLUS_PB6
  - LL_SYSCFG_FASTMODEPLUS_PB7
  - LL_SYSCFG_FASTMODEPLUS_PB8
  - LL_SYSCFG_FASTMODEPLUS_PB9

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 PBx_FMP LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_EnableIT_FPU_IOC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IOC (void)

Function description

Enable Floating Point Unit Invalid operation Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_0 LL_SYSCFG_EnableIT_FPU_IOC

LL_SYSCFG_EnableIT_FPU_DZC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_DZC (void)

Function description

Enable Floating Point Unit Divide-by-zero Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_1 LL_SYSCFG_EnableIT_FPU_DZC

LL_SYSCFG_EnableIT_FPU_UFC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_UFC (void)

Function description

Enable Floating Point Unit Underflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_2 LL_SYSCFG_EnableIT_FPU_UFC

LL_SYSCFG_EnableIT_FPU_OFC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_OFC (void)

Function description

Enable Floating Point Unit Overflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_3 LL_SYSCFG_EnableIT_FPU_OFC

LL_SYSCFG_EnableIT_FPU_IDC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IDC (void)

Function description

Enable Floating Point Unit Input denormal Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_4 LL_SYSCFG_EnableIT_FPU_IDC

LL_SYSCFG_EnableIT_FPU_IXC

Function name

__STATIC_INLINE void LL_SYSCFG_EnableIT_FPU_IXC (void)

Function description

Enable Floating Point Unit Inexact Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_5 LL_SYSCFG_EnableIT_FPU_IXC

LL_SYSCFG_DisableIT_FPU_IOC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IOC (void)

Function description

Disable Floating Point Unit Invalid operation Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_0 LL_SYSCFG_DisableIT_FPU_IOC

LL_SYSCFG_DisableIT_FPU_DZC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_DZC (void)

Function description

Disable Floating Point Unit Divide-by-zero Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_1 LL_SYSCFG_DisableIT_FPU_DZC

LL_SYSCFG_DisableIT_FPU_UFC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_UFC (void)

Function description

Disable Floating Point Unit Underflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_2 LL_SYSCFG_DisableIT_FPU_UFC

LL_SYSCFG_DisableIT_FPU_OFC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_OFC (void)

Function description

Disable Floating Point Unit Overflow Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_3 LL_SYSCFG_DisableIT_FPU_OFC

LL_SYSCFG_DisableIT_FPU_IDC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IDC (void)

Function description

Disable Floating Point Unit Input denormal Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_4 LL_SYSCFG_DisableIT_FPU_IDC

LL_SYSCFG_DisableIT_FPU_IXC

Function name

__STATIC_INLINE void LL_SYSCFG_DisableIT_FPU_IXC (void)

Function description

Disable Floating Point Unit Inexact Interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_5 LL_SYSCFG_DisableIT_FPU_IXC

LL_SYSCFG_IsEnabledIT_FPU_IOC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IOC (void)

Function description

Check if Floating Point Unit Invalid operation Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_0 LL_SYSCFG_IsEnabledIT_FPU_IOC

LL_SYSCFG_IsEnabledIT_FPU_DZC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_DZC (void)

Function description

Check if Floating Point Unit Divide-by-zero Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_1 LL_SYSCFG_IsEnabledIT_FPU_DZC

LL_SYSCFG_IsEnabledIT_FPU_UFC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_UFC (void)

Function description

Check if Floating Point Unit Underflow Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_2 LL_SYSCFG_IsEnabledIT_FPU_UFC

LL_SYSCFG_IsEnabledIT_FPU_OFC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_OFC (void)

Function description

Check if Floating Point Unit Overflow Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_3 LL_SYSCFG_IsEnabledIT_FPU_OFC

LL_SYSCFG_IsEnabledIT_FPU_IDC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IDC (void)

Function description

Check if Floating Point Unit Input denormal Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_4 LL_SYSCFG_IsEnabledIT_FPU_IDC

LL_SYSCFG_IsEnabledIT_FPU_IXC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabledIT_FPU_IXC (void)

Function description

Check if Floating Point Unit Inexact Interrupt source is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_FPUIMR FPU_IE_5 LL_SYSCFG_IsEnabledIT_FPU_IXC

LL_SYSCFG_SetTIMBreakInputs

Function name

__STATIC_INLINE void LL_SYSCFG_SetTIMBreakInputs (uint32_t Break)

Function description

Set connections to TIM1/8/15/16/17 Break inputs.

Parameters

- **Break:** This parameter can be a combination of the following values:
  – LL_SYSCFG_TIMBREAK_ECC
  – LL_SYSCFG_TIMBREAK_PVD
  – LL_SYSCFG_TIMBREAK_SRAM_ECC_LOCK
  – LL_SYSCFG_TIMBREAK_LOCKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 CLL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR2 SPL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR2 PVDL LL_SYSCFG_SetTIMBreakInputs
- SYSCFG_CFGR2 ECCL LL_SYSCFG_SetTIMBreakInputs

LL_SYSCFG_GetTIMBreakInputs

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetTIMBreakInputs (void)

Function description

Get connections to TIM1/8/15/16/17 Break inputs.

Return values

- **Returned:** value can be can be a combination of the following values:
  – LL_SYSCFG_TIMBREAK_ECC
  – LL_SYSCFG_TIMBREAK_PVD
  – LL_SYSCFG_TIMBREAK_SRAM_ECC_LOCK
  – LL_SYSCFG_TIMBREAK_LOCKUP

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 CLL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR2 SPL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR2 PVDL LL_SYSCFG_GetTIMBreakInputs
- SYSCFG_CFGR2 ECCL LL_SYSCFG_GetTIMBreakInputs

LL_SYSCFG_ConfigSecure

Function name

__STATIC_INLINE void LL_SYSCFG_ConfigSecure (uint32_t Configuration)

Function description

Configure Secure mode.

Parameters

- **Configuration:** This parameter shall be the full combination of the following values:
  – LL_SYSCFG_CLOCK_SEC or LL_SYSCFG_CLOCK_NSEC
  – LL_SYSCFG_CLASSB_SEC or LL_SYSCFG_CLASSB_NSEC
  – LL_SYSCFG_FPU_SEC or LL_SYSCFG_FPU_NSEC

Return values

- **None:**

Notes

- Only available from secure state when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR SYSCFGSEC LL_SYSCFG_ConfigSecure
- SECCFGR CLASSBSEC LL_SYSCFG_ConfigSecure
- SECCFGR FPUSEC LL_SYSCFG_ConfigSecure

LL_SYSCFG_GetConfigSecure

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetConfigSecure (void)

Function description

Get Secure mode configuration.

Return values

- **Returned:** value is the combination of the following values:
  - LL_SYSCFG_CLOCK_SEC or LL_SYSCFG_CLOCK_NSEC
  - LL_SYSCFG_CLASSB_SEC or LL_SYSCFG_CLASSB_NSEC
  - LL_SYSCFG_FPU_SEC or LL_SYSCFG_FPU_NSEC

Notes

- Only available when system implements security (TZEN=1)

Reference Manual to LL API cross reference:

- SECCFGR SYSCFGSEC LL_SYSCFG_ConfigSecure
- SECCFGR CLASSBSEC LL_SYSCFG_ConfigSecure
- SECCFGR FPUSEC LL_SYSCFG_ConfigSecure

LL_SYSCFG_GetPMOSVddCompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSVddCompensationValue (void)

Function description

Get the compensation cell value of the GPIO PMOS transistor supplied by VDD.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCVR PCV1 LL_SYSCFG_GetPMOSVddCompensationValue

LL_SYSCFG_GetNMOSVddCompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSVddCompensationValue (void)

Function description

Get the compensation cell value of the GPIO NMOS transistor supplied by VDD.

Return values

- **Returned:** value is the NMOS compensation cell

Reference Manual to LL API cross reference:

- CCVR NCV1 LL_SYSCFG_GetNMOSVddCompensationValue

LL_SYSCFG_GetPMOSVddIO2CompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSVddIO2CompensationValue (void)

Function description

Get the compensation cell value of the GPIO PMOS transistor supplied by VDDIO2.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCVR PCV2 LL_SYSCFG_GetPMOSVddIO2CompensationValue

LL_SYSCFG_GetNMOSVddIO2CompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSVddIO2CompensationValue (void)

Function description

Get the compensation cell value of the GPIO NMOS transistor supplied by VDDIO2.

Return values

- **Returned:** value is the NMOS compensation cell

Reference Manual to LL API cross reference:

- CCVR NCV2 LL_SYSCFG_GetNMOSVddIO2CompensationValue

LL_SYSCFG_GetPMOSVddHSPICompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSVddHSPICompensationValue (void)

Function description

Get the compensation cell value of the HSPI IO PMOS transistor supplied by VDD.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCVR PCV3 LL_SYSCFG_GetPMOSVddHSPICompensationValue

LL_SYSCFG_GetNMOSVddHSPICompensationValue

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSVddHSPICompensationValue (void)

Function description

Get the compensation cell value of the HSPI IO NMOS transistor supplied by VDD.

Return values

- **Returned:** value is the NMOS compensation cell

Reference Manual to LL API cross reference:

- CCVR NCV3 LL_SYSCFG_GetNMOSVddHSPICompensationValue

LL_SYSCFG_SetPMOSVddCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetPMOSVddCompensationCode (uint32_t PMOSCode)

Function description

Set the compensation cell code of the GPIO PMOS transistor supplied by VDD.

Parameters

- **PMOSCode:** PMOS compensation code This code is applied to the PMOS compensation cell when the CS1 bit of the SYSCFG_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR PCC1 LL_SYSCFG_SetPMOSVddCompensationCode

LL_SYSCFG_GetPMOSVddCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSVddCompensationCode (void)

Function description

Get the compensation cell code of the GPIO PMOS transistor supplied by VDD.

Return values

- **Returned:** value is the PMOS compensation cell

Reference Manual to LL API cross reference:

- CCCR PCC1 LL_SYSCFG_GetPMOSVddCompensationCode

LL_SYSCFG_SetPMOSVddIO2CompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetPMOSVddIO2CompensationCode (uint32_t PMOSCode)

Function description

Set the compensation cell code of the GPIO PMOS transistor supplied by VDDIO2.

Parameters

- **PMOSCode:** PMOS compensation code This code is applied to the PMOS compensation cell when the CS2 bit of the SYSCFG_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR PCC2 LL_SYSCFG_SetPMOSVddIO2CompensationCode

LL_SYSCFG_GetPMOSVddIO2CompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSVddIO2CompensationCode (void)

Function description

Get the compensation cell code of the GPIO PMOS transistor supplied by VDDIO2.

Return values

- **Returned:** value is the PMOS compensation

Reference Manual to LL API cross reference:

- CCCR PCC2 LL_SYSCFG_GetPMOSVddIO2CompensationCode

LL_SYSCFG_SetPMOSVddHSPICompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetPMOSVddHSPICompensationCode (uint32_t PMOSCode)

Function description

Set the compensation cell code of the HSPI IO PMOS transistor supplied by VDD.

Parameters

- **PMOSCode:** PMOS compensation code This code is applied to the PMOS compensation cell when the CS3 bit of the SYSCFG_CCCSR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR PCC3 LL_SYSCFG_SetPMOSVddHSPICompensationCode

LL_SYSCFG_GetPMOSVddHSPICompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetPMOSVddHSPICompensationCode (void)

Function description

Get the compensation cell code of the HSPI IO PMOS transistor supplied by VDD.

Return values

- **Returned:** value is the PMOS compensation

Reference Manual to LL API cross reference:

- CCCR PCC3 LL_SYSCFG_GetPMOSVddHSPICompensationCode

LL_SYSCFG_SetNMOSVddCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetNMOSVddCompensationCode (uint32_t NMOSCode)

Function description

Set the compensation cell code of the GPIO NMOS transistor supplied by VDD.

Parameters

- **NMOSCode:** NMOS compensation code This code is applied to the NMOS compensation cell when the CS2 bit of the SYSCFG_CMPCR is set

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR PCC2 LL_SYSCFG_SetNMOSVddCompensationCode

LL_SYSCFG_GetNMOSVddCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSVddCompensationCode (void)

Function description

Get the compensation cell code of the GPIO NMOS transistor supplied by VDD.

Return values

- **Returned:** value is the Vdd compensation cell code for NMOS transistors

Reference Manual to LL API cross reference:

- CCCR NCC1 LL_SYSCFG_GetNMOSVddCompensationCode

LL_SYSCFG_SetNMOSVddIO2CompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetNMOSVddIO2CompensationCode (uint32_t NMOSCode)

Function description

Set the compensation cell code of the GPIO NMOS transistor supplied by VDDIO2.

Parameters

- **NMOSCode:** NMOS compensation code This code is applied to the NMOS compensation cell when the CS2 bit of the SYSCFG_CMPCR is set Value between 0 and 15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR NCC2 LL_SYSCFG_SetNMOSVddIO2CompensationCode

LL_SYSCFG_GetNMOSVddIO2CompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSVddIO2CompensationCode (void)

Function description

Get the compensation cell code of the GPIO NMOS transistor supplied by VDDIO2.

Return values

- **Returned:** value is the NMOS compensation cell code

Reference Manual to LL API cross reference:

- CCCR NCC2 LL_SYSCFG_GetNMOSVddIO2CompensationCode

LL_SYSCFG_SetNMOSVddHSPICompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetNMOSVddHSPICompensationCode (uint32_t NMOSCode)

Function description

Set the compensation cell code of the HSPI IO NMOS transistor supplied by VDD.

Parameters

- **NMOSCode:** NMOS compensation code This code is applied to the NMOS compensation cell when the CS3 bit of the SYSCFG_CCCSR is set Value between 0 and 15

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCR NCC3 LL_SYSCFG_SetNMOSVddHSPICompensationCode

LL_SYSCFG_GetNMOSVddHSPICompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetNMOSVddHSPICompensationCode (void)

Function description

Get the compensation cell code of the HSPI IO NMOS transistor supplied by VDD.

Return values

- **Returned:** value is the NMOS compensation cell code

Reference Manual to LL API cross reference:

- CCCR NCC3 LL_SYSCFG_GetNMOSVddHSPICompensationCode

LL_SYSCFG_EnableVddCompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_EnableVddCompensationCell (void)

Function description

Enable the Compensation Cell of GPIO supplied by VDD.

Return values

- **None:**

Notes

- The vdd compensation cell can be used only when the device supply voltage ranges from 1.71 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN1 LL_SYSCFG_EnableVddCompensationCell

LL_SYSCFG_EnableVddIO2CompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_EnableVddIO2CompensationCell (void)

Function description

Enable the Compensation Cell of GPIO supplied by VDDIO2.

Return values

- **None:**

Notes

- The Vdd I/O compensation cell can be used only when the device supply voltage ranges from 1.08 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN2 LL_SYSCFG_EnableVddIO2CompensationCell

LL_SYSCFG_EnableVddHSPICompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_EnableVddHSPICompensationCell (void)

Function description

Enable the Compensation Cell of HSPI IO supplied by VDD.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR EN3 LL_SYSCFG_EnableVddHSPICompensationCell

LL_SYSCFG_DisableVddCompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_DisableVddCompensationCell (void)

Function description

Disable the Compensation Cell of GPIO supplied by VDD.

Return values

- **None:**

Notes

- The Vdd compensation cell can be used only when the device supply voltage ranges from 1.71 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN1 LL_SYSCFG_DisableVddCompensationCell

LL_SYSCFG_DisableVddIO2CompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_DisableVddIO2CompensationCell (void)

Function description

Disable the Compensation Cell of GPIO supplied by VDDIO2.

Return values

- **None:**

Notes

- The Vdd I/O compensation cell can be used only when the device supply voltage ranges from 1.08 to 3.6 V

Reference Manual to LL API cross reference:

- CCCSR EN2 LL_SYSCFG_DisableVddIO2CompensationCell

LL_SYSCFG_DisableVddHSPICompensationCell

Function name

__STATIC_INLINE void LL_SYSCFG_DisableVddHSPICompensationCell (void)

Function description

Disable the Compensation Cell of HSPI IO supplied by VDD.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR EN3 LL_SYSCFG_DisableVddHSPICompensationCell

LL_SYSCFG_IsEnabled_VddCompensationCell

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabled_VddCompensationCell (void)

Function description

Check if the Compensation Cell of GPIO supplied by VDD is enable.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR EN1 LL_SYSCFG_IsEnabled_VddCompensationCell

LL_SYSCFG_IsEnabled_VddIO2CompensationCell

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabled_VddIO2CompensationCell (void)

Function description

Check if the Compensation Cell of GPIO supplied by VDDIO2 is enable.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR EN2 LL_SYSCFG_IsEnabled_VddIO2CompensationCell

LL_SYSCFG_IsEnabled_VddHSPICompensationCell

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsEnabled_VddHSPICompensationCell (void)

Function description

Check if the Compensation Cell of HSPI IO supplied by VDD is enable.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR EN3 LL_SYSCFG_IsEnabled_VddHSPICompensationCell

LL_SYSCFG_IsActiveFlag_VddCMPCR

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_VddCMPCR (void)

Function description

Get Compensation Cell ready Flag of GPIO supplied by VDD.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR RDY1 LL_SYSCFG_IsActiveFlag_VddCMPCR

LL_SYSCFG_IsActiveFlag_VddIO2CMPCR

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_VddIO2CMPCR (void)

Function description

Get Compensation Cell ready Flag of GPIO supplied by VDDIO2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR RDY2 LL_SYSCFG_IsActiveFlag_VddIO2CMPCR

LL_SYSCFG_IsActiveFlag_VddHSPICMPCR

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_VddHSPICMPCR (void)

Function description

Get Compensation Cell ready Flag of HSPI IO supplied by VDD.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCCSR RDY3 LL_SYSCFG_IsActiveFlag_VddHSPICMPCR

LL_SYSCFG_SetVddCellCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetVddCellCompensationCode (uint32_t CompCode)

Function description

Set the compensation cell code selection of GPIO supplied by VDD.

Parameters

- **CompCode:** Selects the code to be applied for the Vdd compensation cell This parameter can be one of the following values:
  - LL_SYSCFG_VDD_CELL_CODE : Select Code from the cell (available in the SYSCFG_CCVR)
  - LL_SYSCFG_VDD_REGISTER_CODE: Select Code from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR CS1 LL_SYSCFG_SetVddCellCompensationCode

LL_SYSCFG_SetVddIO2CellCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetVddIO2CellCompensationCode (uint32_t CompCode)

Function description

Set the compensation cell code selection of GPIO supplied by VDDIO2.

Parameters

- **CompCode:** Selects the code to be applied for the VddIO compensation cell This parameter can be one of the following values:
  - LL_SYSCFG_VDDIO2_CELL_CODE : Select Code from the cell (available in the SYSCFG_CCVR)
  - LL_SYSCFG_VDDIO2_REGISTER_CODE: Select Code from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR CS2 LL_SYSCFG_SetVddIO2CellCompensationCode

LL_SYSCFG_SetVddHSPICellCompensationCode

Function name

__STATIC_INLINE void LL_SYSCFG_SetVddHSPICellCompensationCode (uint32_t CompCode)

Function description

Set the compensation cell code selection of HSPI IO supplied by VDD.

Parameters

- **CompCode:** Selects the code to be applied for the VddIO compensation cell This parameter can be one of the following values:
  - LL_SYSCFG_VDDHSPI_CELL_CODE : Select Code from the cell (available in the SYSCFG_CCVR)
  - LL_SYSCFG_VDDHSPI_REGISTER_CODE: Select Code from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCCSR CS3 LL_SYSCFG_SetVddHSPICellCompensationCode

LL_SYSCFG_GetVddCellCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetVddCellCompensationCode (void)

Function description

Get the compensation cell code selection of GPIO supplied by VDD.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_VDD_CELL_CODE : Selected Code is from the cell (available in the SYSCFG_CCVR)
  - LL_SYSCFG_VDD_REGISTER_CODE: Selected Code is from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Reference Manual to LL API cross reference:

- CCCSR CS1 LL_SYSCFG_GetVddCellCompensationCode

LL_SYSCFG_GetVddIO2CellCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetVddIO2CellCompensationCode (void)

Function description

Get the compensation cell code selection of GPIO supplied by VDDIO2.

Return values

- **Returned:** value can be one of the following values:
  – LL_SYSCFG_VDDIO2_CELL_CODE : Selected Code is from the cell (available in the SYSCFG_CCVR)
  – LL_SYSCFG_VDDIO2_REGISTER_CODE: Selected Code is from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Reference Manual to LL API cross reference:

- CCCSR CS2 LL_SYSCFG_GetVddIO2CellCompensationCode

LL_SYSCFG_GetVddHSPICellCompensationCode

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetVddHSPICellCompensationCode (void)

Function description

Get the compensation cell code selection of HSPI IO supplied by VDD.

Return values

- **Returned:** value can be one of the following values:
  – LL_SYSCFG_VDDHSPI_CELL_CODE : Selected Code is from the cell (available in the SYSCFG_CCVR)
  – LL_SYSCFG_VDDHSPI_REGISTER_CODE: Selected Code is from the SYSCFG compensation cell code register (SYSCFG_CCCR)

Reference Manual to LL API cross reference:

- CCCSR CS3 LL_SYSCFG_GetVddHSPICellCompensationCode

LL_DBGMCU_GetDeviceID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetDeviceID (void)

Function description

Return the device identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF (ex: device ID is 0x6415)

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE DEV_ID LL_DBGMCU_GetDeviceID

LL_DBGMCU_GetRevisionID

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetRevisionID (void)

Function description

Return the device revision identifier.

Return values

- **Values:** between Min_Data=0x00 and Max_Data=0xFFFF

Notes

- This field indicates the revision of the device.

Reference Manual to LL API cross reference:

- DBGMCU_IDCODE REV_ID LL_DBGMCU_GetRevisionID

LL_DBGMCU_EnableDBGStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGStopMode (void)

Function description

Enable the Debug Module during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STOP LL_DBGMCU_EnableDBGStopMode

LL_DBGMCU_DisableDBGStopMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STOP LL_DBGMCU_DisableDBGStopMode

LL_DBGMCU_EnableDBGStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_EnableDBGStandbyMode (void)

Function description

Enable the Debug Module during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_EnableDBGStandbyMode

LL_DBGMCU_DisableDBGStandbyMode

Function name

__STATIC_INLINE void LL_DBGMCU_DisableDBGStandbyMode (void)

Function description

Disable the Debug Module during STANDBY mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

LL_DBGMCU_EnableTraceClock

Function name

__STATIC_INLINE void LL_DBGMCU_EnableTraceClock (void)

Function description

Enable the Debug Clock Trace.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_CLKEN LL_DBGMCU_EnableTraceClock

LL_DBGMCU_DisableTraceClock

Function name

__STATIC_INLINE void LL_DBGMCU_DisableTraceClock (void)

Function description

Disable the Debug Clock Trace.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_CLKEN LL_DBGMCU_DisableTraceClock

LL_DBGMCU_IsEnabledTraceClock

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_IsEnabledTraceClock (void)

Function description

Check if clock trace is enabled or disabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DBGMCU_CR_TRACE_CLKEN LL_DBGMCU_IsEnabledTraceClock

LL_DBGMCU_SetTracePinAssignment

Function name

__STATIC_INLINE void LL_DBGMCU_SetTracePinAssignment (uint32_t PinAssignment)

Function description

Set Trace pin assignment control.

Parameters

- **PinAssignment:** This parameter can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_SetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_SetTracePinAssignment

LL_DBGMCU_GetTracePinAssignment

Function name

__STATIC_INLINE uint32_t LL_DBGMCU_GetTracePinAssignment (void)

Function description

Get Trace pin assignment control.

Return values

- **Returned:** value can be one of the following values:
  - LL_DBGMCU_TRACE_NONE
  - LL_DBGMCU_TRACE_ASYNCH
  - LL_DBGMCU_TRACE_SYNCH_SIZE1
  - LL_DBGMCU_TRACE_SYNCH_SIZE2
  - LL_DBGMCU_TRACE_SYNCH_SIZE4

Reference Manual to LL API cross reference:

- DBGMCU_CR TRACE_IOEN LL_DBGMCU_GetTracePinAssignment
- DBGMCU_CR TRACE_MODE LL_DBGMCU_GetTracePinAssignment

LL_DBGMCU_APB1_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph

LL_DBGMCU_APB1_GRP2_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB1 peripherals (group2 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_I2C4_STOP
  - LL_DBGMCU_APB1_GRP2_LPTIM2_STOP
  - LL_DBGMCU_APB1_GRP2_I2C5_STOP (*)
  - LL_DBGMCU_APB1_GRP2_I2C6_STOP (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices.

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph

LL_DBGMCU_APB1_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group1 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP1_TIM2_STOP
  - LL_DBGMCU_APB1_GRP1_TIM3_STOP
  - LL_DBGMCU_APB1_GRP1_TIM4_STOP
  - LL_DBGMCU_APB1_GRP1_TIM5_STOP
  - LL_DBGMCU_APB1_GRP1_TIM6_STOP
  - LL_DBGMCU_APB1_GRP1_TIM7_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP
  - LL_DBGMCU_APB1_GRP1_I2C2_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR1 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB1 peripherals (group2 peripherals)

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_I2C4_STOP
  - LL_DBGMCU_APB1_GRP2_LPTIM2_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB1FZR2 DBG_xxxx_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph

LL_DBGMCU_APB2_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB2_GRP1_TIM1_STOP
  - LL_DBGMCU_APB2_GRP1_TIM8_STOP
  - LL_DBGMCU_APB2_GRP1_TIM15_STOP
  - LL_DBGMCU_APB2_GRP1_TIM16_STOP
  - LL_DBGMCU_APB2_GRP1_TIM17_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZ DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB2_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_DBGMCU_APB2_GRP1_TIM1_STOP
  – LL_DBGMCU_APB2_GRP1_TIM8_STOP
  – LL_DBGMCU_APB2_GRP1_TIM15_STOP
  – LL_DBGMCU_APB2_GRP1_TIM16_STOP
  – LL_DBGMCU_APB2_GRP1_TIM17_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB2FZR DBG_TIMx_STOP LL_DBGMCU_APB2_GRP1_UnFreezePeriph

LL_DBGMCU_APB3_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB3_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze APB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_DBGMCU_APB3_GRP1_I2C3_STOP
  – LL_DBGMCU_APB3_GRP1_LPTIM1_STOP
  – LL_DBGMCU_APB3_GRP1_LPTIM3_STOP
  – LL_DBGMCU_APB3_GRP1_LPTIM4_STOP
  – LL_DBGMCU_APB3_GRP1_RTC_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB3FZ DBG_TIMx_STOP LL_DBGMCU_APB3_GRP1_FreezePeriph

LL_DBGMCU_APB3_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB3_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB3_GRP1_I2C3_STOP
  - LL_DBGMCU_APB3_GRP1_LPTIM1_STOP
  - LL_DBGMCU_APB3_GRP1_LPTIM3_STOP
  - LL_DBGMCU_APB3_GRP1_LPTIM4_STOP
  - LL_DBGMCU_APB3_GRP1_RTC_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_APB3FZR DBG_TIMx_STOP LL_DBGMCU_APB3_GRP1_UnFreezePeriph

LL_DBGMCU_AHB1_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_AHB1_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze AHB1 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_AHB1_GRP1_GPDMA0_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA1_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA2_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA3_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA4_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA5_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA6_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA7_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA8_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA9_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA10_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA11_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA12_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA13_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA14_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA15_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_AHB1FZ DBG_GPDMAx_STOP LL_DBGMCU_AHB1_GRP1_FreezePeriph

LL_DBGMCU_AHB1_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_AHB1_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze AHB1 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_AHB1_GRP1_GPDMA0_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA1_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA2_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA3_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA4_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA5_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA6_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA7_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA8_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA9_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA10_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA11_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA12_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA13_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA14_STOP
  - LL_DBGMCU_AHB1_GRP1_GPDMA15_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_AHB1FZ DBG_GPDMAx_STOP LL_DBGMCU_AHB1_GRP1_FreezePeriph

LL_DBGMCU_AHB3_GRP1_FreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_AHB3_GRP1_FreezePeriph (uint32_t Periphs)

Function description

Freeze AHB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_AHB3_GRP1_LPDMA0_STOP
  - LL_DBGMCU_AHB3_GRP1_LPDMA1_STOP
  - LL_DBGMCU_AHB3_GRP1_LPDMA2_STOP
  - LL_DBGMCU_AHB3_GRP1_LPDMA3_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_AHB3FZ DBG_LPDMAx_STOP LL_DBGMCU_AHB3_GRP1_FreezePeriph

LL_DBGMCU_AHB3_GRP1_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_AHB3_GRP1_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze AHB3 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_DBGMCU_AHB3_GRP1_LPDMA0_STOP
  – LL_DBGMCU_AHB3_GRP1_LPDMA1_STOP
  – LL_DBGMCU_AHB3_GRP1_LPDMA2_STOP
  – LL_DBGMCU_AHB3_GRP1_LPDMA3_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGMCU_AHB3FZ DBG_LPDMAx_STOP LL_DBGMCU_AHB3_GRP1_FreezePeriph

LL_VREFBUF_Enable

Function name

__STATIC_INLINE void LL_VREFBUF_Enable (void)

Function description

Enable Internal voltage reference.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR ENVR LL_VREFBUF_Enable

LL_VREFBUF_Disable

Function name

__STATIC_INLINE void LL_VREFBUF_Disable (void)

Function description

Disable Internal voltage reference.

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR ENVR LL_VREFBUF_Disable

LL_VREFBUF_EnableHIZ

Function name

__STATIC_INLINE void LL_VREFBUF_EnableHIZ (void)

Function description

Enable high impedance (VREF+pin is high impedance)

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR HIZ LL_VREFBUF_EnableHIZ

LL_VREFBUF_DisableHIZ

Function name

__STATIC_INLINE void LL_VREFBUF_DisableHIZ (void)

Function description

Disable high impedance (VREF+pin is internally connected to the voltage reference buffer output)

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR HIZ LL_VREFBUF_DisableHIZ

LL_VREFBUF_SetVoltageScaling

Function name

__STATIC_INLINE void LL_VREFBUF_SetVoltageScaling (uint32_t Scale)

Function description

Set the Voltage reference scale.

Parameters

- **Scale:** This parameter can be one of the following values:
  – LL_VREFBUF_VOLTAGE_SCALE0
  – LL_VREFBUF_VOLTAGE_SCALE1
  – LL_VREFBUF_VOLTAGE_SCALE2
  – LL_VREFBUF_VOLTAGE_SCALE3

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRS LL_VREFBUF_SetVoltageScaling

LL_VREFBUF_GetVoltageScaling

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_GetVoltageScaling (void)

Function description

Get the Voltage reference scale.

Return values

- **Returned:** value can be one of the following values:
  – LL_VREFBUF_VOLTAGE_SCALE0
  – LL_VREFBUF_VOLTAGE_SCALE1
  – LL_VREFBUF_VOLTAGE_SCALE2
  – LL_VREFBUF_VOLTAGE_SCALE3

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRS LL_VREFBUF_GetVoltageScaling

LL_VREFBUF_IsVREFReady

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_IsVREFReady (void)

Function description

Check if Voltage reference buffer is ready.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- VREFBUF_CSR VRR LL_VREFBUF_IsVREFReady

LL_VREFBUF_GetTrimming

Function name

__STATIC_INLINE uint32_t LL_VREFBUF_GetTrimming (void)

Function description

Get the trimming code for VREFBUF calibration.

Return values

- **Between:** 0 and 0x3F

Reference Manual to LL API cross reference:

- VREFBUF_CCR TRIM LL_VREFBUF_GetTrimming

LL_VREFBUF_SetTrimming

Function name

__STATIC_INLINE void LL_VREFBUF_SetTrimming (uint32_t Value)

Function description

Set the trimming code for VREFBUF calibration (Tune the internal reference buffer voltage)

Parameters

- **Value:** Between 0 and 0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- VREFBUF_CCR TRIM LL_VREFBUF_SetTrimming

LL_FLASH_SetLatency

Function name

__STATIC_INLINE void LL_FLASH_SetLatency (uint32_t Latency)

Function description

Set FLASH Latency.

Parameters

- **Latency:** This parameter can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5
  - LL_FLASH_LATENCY_6
  - LL_FLASH_LATENCY_7
  - LL_FLASH_LATENCY_8
  - LL_FLASH_LATENCY_9
  - LL_FLASH_LATENCY_10
  - LL_FLASH_LATENCY_11
  - LL_FLASH_LATENCY_12
  - LL_FLASH_LATENCY_13
  - LL_FLASH_LATENCY_14
  - LL_FLASH_LATENCY_15

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_SetLatency

LL_FLASH_GetLatency

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetLatency (void)

Function description

Get FLASH Latency.

Return values

- **Returned:** value can be one of the following values:
  - LL_FLASH_LATENCY_0
  - LL_FLASH_LATENCY_1
  - LL_FLASH_LATENCY_2
  - LL_FLASH_LATENCY_3
  - LL_FLASH_LATENCY_4
  - LL_FLASH_LATENCY_5
  - LL_FLASH_LATENCY_6
  - LL_FLASH_LATENCY_7
  - LL_FLASH_LATENCY_8
  - LL_FLASH_LATENCY_9
  - LL_FLASH_LATENCY_10
  - LL_FLASH_LATENCY_11
  - LL_FLASH_LATENCY_12
  - LL_FLASH_LATENCY_13
  - LL_FLASH_LATENCY_14
  - LL_FLASH_LATENCY_15

Reference Manual to LL API cross reference:

- FLASH_ACR LATENCY LL_FLASH_GetLatency

LL_FLASH_EnableRunPowerDown

Function name

__STATIC_INLINE void LL_FLASH_EnableRunPowerDown (void)

Function description

Enable Flash Power-down mode during run mode or Low-power run mode.

Return values

- **None:**

Notes

- Flash memory can be put in power-down mode only when the code is executed from RAM
- Flash must not be accessed when power down is enabled
- Flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to LL API cross reference:

- FLASH_ACR RUN_PD LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY1_1 LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY1_2 LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY2_1 LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY2_2 LL_FLASH_EnableRunPowerDown

LL_FLASH_EnableRunPowerDownBank1

Function name

__STATIC_INLINE void LL_FLASH_EnableRunPowerDownBank1 (void)

Function description

Enable flash Power-down mode during run mode or Low-power run mode of bank1.

Return values

- **None:**

Notes

- Bank 1 of flash memory can be put in power-down mode only when the code is executed from RAM
- Bank1 of flash must not be accessed when power down is enabled
- Bank1 of flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to LL API cross reference:

- FLASH_ACR RUN_PD LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY1_1 LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY1_2 LL_FLASH_EnableRunPowerDown
-

LL_FLASH_EnableRunPowerDownBank2

Function name

__STATIC_INLINE void LL_FLASH_EnableRunPowerDownBank2 (void)

Function description

Enable flash Power-down mode during run mode or Low-power run mode of Bank2.

Return values

- **None:**

Notes

- Bank 2 of flash memory can be put in power-down mode only when the code is executed from RAM
- Bank2 of flash must not be accessed when power down is enabled
- Bank2 of flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to LL API cross reference:

- FLASH_ACR RUN_PD LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY2_1 LL_FLASH_EnableRunPowerDown
- FLASH_PDKEYR PDKEY2_2 LL_FLASH_EnableRunPowerDown
-

LL_FLASH_EnableSleepPowerDown

Function name

__STATIC_INLINE void LL_FLASH_EnableSleepPowerDown (void)

Function description

Enable Flash Power-down mode during Sleep or Low-power sleep mode.

Return values

- **None:**

Notes

- Flash must not be put in power-down while a program or an erase operation is on-going

Reference Manual to LL API cross reference:

- FLASH_ACR SLEEP_PD LL_FLASH_EnableSleepPowerDown

LL_FLASH_DisableSleepPowerDown

Function name

__STATIC_INLINE void LL_FLASH_DisableSleepPowerDown (void)

Function description

Disable Flash Power-down mode during Sleep or Low-power sleep mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR SLEEP_PD LL_FLASH_DisableSleepPowerDown

LL_SYSCFG_ClearEraseEndStatus

Function name

__STATIC_INLINE void LL_SYSCFG_ClearEraseEndStatus (void)

Function description

Clear Status of End of Erase for ICACHE and PKA RAMs.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MESR IPMEE LL_SYSCFG_ClearEraseEndStatus

LL_SYSCFG_GetEraseEndStatus

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetEraseEndStatus (void)

Function description

Get Status of End of Erase for ICACHE and PKA RAMs.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_MEMORIES_ERASE_ON_GOING : Erase of memories not yet done
  - LL_SYSCFG_MEMORIES_ERASE_ENDED: Erase of memories ended

Reference Manual to LL API cross reference:

- MESR IPMEE LL_SYSCFG_GetEraseEndStatus

LL_SYSCFG_ClearEraseAfterResetStatus

Function name

__STATIC_INLINE void LL_SYSCFG_ClearEraseAfterResetStatus (void)

Function description

Clear Status of End of Erase after reset for SRAM2, BKPRAM, ICACHE, DCACHE,PKA rams.

Return values

- **None:**

Reference Manual to LL API cross reference:

- MESR MCLR LL_SYSCFG_ClearEraseAfterResetStatus

LL_SYSCFG_GetEraseAfterResetStatus

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetEraseAfterResetStatus (void)

Function description

Get Status of End of Erase after reset for SRAM2, BKPRAM, ICACHE, DCACHE,PKA rams.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_MEMORIES_ERASE_ON_GOING : Erase of memories not yet done
  - LL_SYSCFG_MEMORIES_ERASE_ENDED: Erase of memories ended

Reference Manual to LL API cross reference:

- MESR MCLR LL_SYSCFG_GetEraseAfterResetStatus

118.2 SYSTEM Firmware driver defines

The following section lists the various define and macros of the module.

118.2.1 SYSTEM

SYSTEM

DBGMCU AHB1 GRP1 STOP IP

LL_DBGMCU_AHB1_GRP1_GPDMA0_STOP
The counter clock of GPDMA0 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA1_STOP
The counter clock of GPDMA1 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA2_STOP
The counter clock of GPDMA2 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA3_STOP
The counter clock of GPDMA3 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA4_STOP
The counter clock of GPDMA4 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA5_STOP
The counter clock of GPDMA5 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA6_STOP
The counter clock of GPDMA6 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA7_STOP
The counter clock of GPDMA7 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA8_STOP
The counter clock of GPDMA8 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA9_STOP
The counter clock of GPDMA9 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA10_STOP
The counter clock of GPDMA10 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA11_STOP
The counter clock of GPDMA11 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA12_STOP
The counter clock of GPDMA12 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA13_STOP
The counter clock of GPDMA13 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA14_STOP
The counter clock of GPDMA14 is stopped when the core is halted

LL_DBGMCU_AHB1_GRP1_GPDMA15_STOP
The counter clock of GPDMA15 is stopped when the core is halted

DBGMCU AHB3 GRP1 STOP IP

LL_DBGMCU_AHB3_GRP1_LPDMA0_STOP
The counter clock of LPDMA0 is stopped when the core is halted

LL_DBGMCU_AHB3_GRP1_LPDMA1_STOP
The counter clock of LPDMA1 is stopped when the core is halted

LL_DBGMCU_AHB3_GRP1_LPDMA2_STOP
The counter clock of LPDMA2 is stopped when the core is halted

LL_DBGMCU_AHB3_GRP1_LPDMA3_STOP

The counter clock of LPDMA3 is stopped when the core is halted

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP

The counter clock of TIM2 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM3_STOP

The counter clock of TIM3 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM4_STOP

The counter clock of TIM4 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM5_STOP

The counter clock of TIM5 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM6_STOP

The counter clock of TIM6 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_TIM7_STOP

The counter clock of TIM7 is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_WWDG_STOP

The window watchdog counter clock is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_IWDG_STOP

The independent watchdog counter clock is stopped when the core is halted

LL_DBGMCU_APB1_GRP1_I2C1_STOP

The I2C1 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP1_I2C2_STOP

The I2C2 SMBus timeout is frozen

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_I2C4_STOP

The I2C4 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP2_LPTIM2_STOP

The counter clock of LPTIM2 is stopped when the core is halted

LL_DBGMCU_APB1_GRP2_I2C5_STOP

The I2C5 SMBus timeout is frozen

LL_DBGMCU_APB1_GRP2_I2C6_STOP

The I2C6 SMBus timeout is frozen

DBGMCU APB2 GRP1 STOP IP

LL_DBGMCU_APB2_GRP1_TIM1_STOP

The counter clock of TIM1 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM8_STOP

The counter clock of TIM8 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM15_STOP

The counter clock of TIM15 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM16_STOP

The counter clock of TIM16 is stopped when the core is halted

LL_DBGMCU_APB2_GRP1_TIM17_STOP

The counter clock of TIM17 is stopped when the core is halted

DBGMCU APB3 GRP1 STOP IP

LL_DBGMCU_APB3_GRP1_I2C3_STOP

The counter clock of I2C3 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM1_STOP

The counter clock of LPTIM1 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM3_STOP

The counter clock of LPTIM3 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_LPTIM4_STOP

The counter clock of LPTIM4 is stopped when the core is halted

LL_DBGMCU_APB3_GRP1_RTC_STOP

The counter clock of RTC is stopped when the core is halted

SYSCFG Vdd compensation cell Code selection

LL_SYSCFG_VDD_CELL_CODE

LL_SYSCFG_VDD_REGISTER_CODE

SYSCFG VddIO2 compensation cell Code selection

LL_SYSCFG_VDDIO2_CELL_CODE

LL_SYSCFG_VDDIO2_REGISTER_CODE

SYSCFG VddHSPI compensation cell Code selection

LL_SYSCFG_VDDHSPI_CELL_CODE

LL_SYSCFG_VDDHSPI_REGISTER_CODE

SYSCFG MEMORIES ERASE STATUS

LL_SYSCFG_MEMORIES_ERASE_ON_GOING

LL_SYSCFG_MEMORIES_ERASE_ENDED

EXTI EXTI LINE

LL_EXTI_EXTI_LINE0

EXTI_POSITION_0 | EXTICR[0]

LL_EXTI_EXTI_LINE1

EXTI_POSITION_8 | EXTICR[0]

LL_EXTI_EXTI_LINE2
EXTI_POSITION_16 | EXTICR[0]

LL_EXTI_EXTI_LINE3
EXTI_POSITION_24 | EXTICR[0]

LL_EXTI_EXTI_LINE4
EXTI_POSITION_0 | EXTICR[1]

LL_EXTI_EXTI_LINE5
EXTI_POSITION_8 | EXTICR[1]

LL_EXTI_EXTI_LINE6
EXTI_POSITION_16 | EXTICR[1]

LL_EXTI_EXTI_LINE7
EXTI_POSITION_24 | EXTICR[1]

LL_EXTI_EXTI_LINE8
EXTI_POSITION_0 | EXTICR[2]

LL_EXTI_EXTI_LINE9
EXTI_POSITION_8 | EXTICR[2]

LL_EXTI_EXTI_LINE10
EXTI_POSITION_16 | EXTICR[2]

LL_EXTI_EXTI_LINE11
EXTI_POSITION_24 | EXTICR[2]

LL_EXTI_EXTI_LINE12
EXTI_POSITION_0 | EXTICR[3]

LL_EXTI_EXTI_LINE13
EXTI_POSITION_8 | EXTICR[3]

LL_EXTI_EXTI_LINE14
EXTI_POSITION_16 | EXTICR[3]

LL_EXTI_EXTI_LINE15
EXTI_POSITION_24 | EXTICR[3]

EXTI EXTI PORT

LL_EXTI_EXTI_PORTA
EXTI PORT A

LL_EXTI_EXTI_PORTB
EXTI PORT B

LL_EXTI_EXTI_PORTC
EXTI PORT C

LL_EXTI_EXTI_PORTD
EXTI PORT D

LL_EXTI_EXTI_PORTE
EXTI PORT E

LL_EXTI_EXTI_PORTF

EXTI PORT F

LL_EXTI_EXTI_PORTG

EXTI PORT G

LL_EXTI_EXTI_PORTH

EXTI PORT H

LL_EXTI_EXTI_PORTI

EXTI PORT I

LL_EXTI_EXTI_PORTJ

EXTI PORT J

SYSCFG FASTMODEPLUS

LL_SYSCFG_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

LL_SYSCFG_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

LL_SYSCFG_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

LL_SYSCFG_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

FLASH LATENCY

LL_FLASH_LATENCY_0

FLASH zero wait state

LL_FLASH_LATENCY_1

FLASH one wait state

LL_FLASH_LATENCY_2

FLASH two wait states

LL_FLASH_LATENCY_3

FLASH three wait states

LL_FLASH_LATENCY_4

FLASH four wait states

LL_FLASH_LATENCY_5

FLASH five wait states

LL_FLASH_LATENCY_6

FLASH six wait state

LL_FLASH_LATENCY_7

FLASH Seven wait states

LL_FLASH_LATENCY_8

FLASH Eight wait states

LL_FLASH_LATENCY_9

FLASH nine wait states

LL_FLASH_LATENCY_10

FLASH ten wait states

LL_FLASH_LATENCY_11

FLASH eleven wait states

LL_FLASH_LATENCY_12

FLASH twelve wait states

LL_FLASH_LATENCY_13

FLASH thirteen wait states

LL_FLASH_LATENCY_14

FLASH fourteen wait states

LL_FLASH_LATENCY_15

FLASH fifteen wait states

Secure attributes

LL_SYSCFG_CLOCK_SEC

SYSCFG clock configuration secure-only access

LL_SYSCFG_CLOCK_NSEC

SYSCFG clock configuration secure/non-secure access

LL_SYSCFG_CLASSB_SEC

Class B configuration secure-only access

LL_SYSCFG_CLASSB_NSEC

Class B configuration secure/non-secure access

LL_SYSCFG_FPU_SEC

FPU configuration secure-only access

LL_SYSCFG_FPU_NSEC

FPU configuration secure/non-secure access

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_ECC

Enables and locks the ECC error signal with Break Input of TIM1/8/15/16/17

LL_SYSCFG_TIMBREAK_PVD

Enables and locks the PVD connection with TIM1/8/15/16/17 Break Input and also the PVDE and PLS bits of the Power Control Interface

LL_SYSCFG_TIMBREAK_SRAM_ECC_LOCK

Enables and locks the SRAM ECC double error signal with Break Input of TIM1/8/15/16/17

LL_SYSCFG_TIMBREAK_LOCKUP

Enables and locks the LOCKUP output of CortexM33 with Break Input of TIM1/15/16/17

DBGMCU TRACE Pin Assignment

LL_DBGMCU_TRACE_NONE

TRACE pins not assigned (default state)

LL_DBGMCU_TRACE_ASYNCH

TRACE pin assignment for Asynchronous Mode

LL_DBGMCU_TRACE_SYNCH_SIZE1

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 1

LL_DBGMCU_TRACE_SYNCH_SIZE2

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 2

LL_DBGMCU_TRACE_SYNCH_SIZE4

TRACE pin assignment for Synchronous Mode with a TRACEDATA size of 4

VREFBUF VOLTAGE

LL_VREFBUF_VOLTAGE_SCALE0

Voltage reference scale 0 (VREF_OUT1)

LL_VREFBUF_VOLTAGE_SCALE1

Voltage reference scale 1 (VREF_OUT2)

LL_VREFBUF_VOLTAGE_SCALE2

Voltage reference scale 0 (VREF_OUT3)

LL_VREFBUF_VOLTAGE_SCALE3

Voltage reference scale 1 (VREF_OUT4)

119 LL TIM Generic Driver

119.1 TIM Firmware driver registers structures

119.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32u5xx_ll_tim.h

Data Fields

- ***uint16_t Prescaler***
- ***uint32_t CounterMode***
- ***uint32_t Autoreload***
- ***uint32_t ClockDivision***
- ***uint32_t RepetitionCounter***

Field Documentation

- ***uint16_t LL_TIM_InitTypeDef::Prescaler***
  Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetPrescaler()**.
- ***uint32_t LL_TIM_InitTypeDef::CounterMode***
  Specifies the counter mode. This parameter can be a value of ***TIM_LL_EC_COUNTERMODE***.This feature can be modified afterwards using unitary function **LL_TIM_SetCounterMode()**.
- ***uint32_t LL_TIM_InitTypeDef::Autoreload***
  Specifies the auto reload value to be loaded into the active Auto-Reload Register at the next update event. This parameter must be a number between Min_Data=0x0000 and Max_Data=0xFFFF. Some timer instances may support 32 bits counters. In that case this parameter must be a number between 0x0000 and 0xFFFFFFFF.This feature can be modified afterwards using unitary function **LL_TIM_SetAutoReload()**.
- ***uint32_t LL_TIM_InitTypeDef::ClockDivision***
  Specifies the clock division. This parameter can be a value of ***TIM_LL_EC_CLOCKDIVISION***.This feature can be modified afterwards using unitary function **LL_TIM_SetClockDivision()**.
- ***uint32_t LL_TIM_InitTypeDef::RepetitionCounter***
  Specifies the repetition counter value. Each time the RCR downcounter reaches zero, an update event is generated and counting restarts from the RCR value (N). This means in PWM mode that (N+1) corresponds to:
  - the number of PWM periods in edge-aligned mode
  - the number of half PWM period in center-aligned mode GP timers: this parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF. Advanced timers: this parameter must be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.

  This feature can be modified afterwards using unitary function **LL_TIM_SetRepetitionCounter()**.

119.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32u5xx_ll_tim.h

Data Fields

- ***uint32_t OCMode***
- ***uint32_t OCState***
- ***uint32_t OCNState***
- ***uint32_t CompareValue***
- ***uint32_t OCPolarity***
- ***uint32_t OCNPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t OCNIdleState***

Field Documentation

- ***uint32_t LL_TIM_OC_InitTypeDef::OCMode***
  Specifies the output mode. This parameter can be a value of ***TIM_LL_EC_OCMODE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetMode()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCState***
  Specifies the TIM Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNState***
  Specifies the TIM complementary Output Compare state. This parameter can be a value of ***TIM_LL_EC_OCSTATE***.This feature can be modified afterwards using unitary functions **LL_TIM_CC_EnableChannel()** or **LL_TIM_CC_DisableChannel()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::CompareValue***
  Specifies the Compare value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data=0x0000 and Max_Data=0xFFFF.This feature can be modified afterwards using unitary function LL_TIM_OC_SetCompareCHx (x=1..6).
- ***uint32_t LL_TIM_OC_InitTypeDef::OCPolarity***
  Specifies the output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNPolarity***
  Specifies the complementary output polarity. This parameter can be a value of ***TIM_LL_EC_OCPOLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetPolarity()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.
- ***uint32_t LL_TIM_OC_InitTypeDef::OCNIdleState***
  Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of ***TIM_LL_EC_OCIDLESTATE***.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetIdleState()**.

119.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32u5xx_ll_tim.h

Data Fields

- ***uint32_t ICPolarity***
- ***uint32_t ICActiveInput***
- ***uint32_t ICPrescaler***
- ***uint32_t ICFilter***

Field Documentation

- ***uint32_t LL_TIM_IC_InitTypeDef::ICPolarity***
  Specifies the active edge of the input signal. This parameter can be a value of ***TIM_LL_EC_IC_POLARITY***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICActiveInput***
  Specifies the input. This parameter can be a value of ***TIM_LL_EC_ACTIVEINPUT***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICPrescaler***
  Specifies the Input Capture Prescaler. This parameter can be a value of ***TIM_LL_EC_ICPSC***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_IC_InitTypeDef::ICFilter***
  Specifies the input capture filter. This parameter can be a value of ***TIM_LL_EC_IC_FILTER***.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

119.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32u5xx_ll_tim.h

Data Fields

- ***uint32_t EncoderMode***
- ***uint32_t IC1Polarity***
- ***uint32_t IC1ActiveInput***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t IC2Polarity***
- ***uint32_t IC2ActiveInput***
- ***uint32_t IC2Prescaler***
- ***uint32_t IC2Filter***

Field Documentation

- ***uint32_t LL_TIM_ENCODER_InitTypeDef::EncoderMode***
  Specifies the encoder resolution (x2 or x4). This parameter can be a value of *TIM_LL_EC_ENCODERMODE*.This feature can be modified afterwards using unitary function **LL_TIM_SetEncoderMode()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of *TIM_LL_EC_IC_POLARITY*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1ActiveInput***
  Specifies the TI1 input source This parameter can be a value of *TIM_LL_EC_ACTIVEINPUT*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Polarity***
  Specifies the active edge of TI2 input. This parameter can be a value of *TIM_LL_EC_IC_POLARITY*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2ActiveInput***
  Specifies the TI2 input source This parameter can be a value of *TIM_LL_EC_ACTIVEINPUT*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetActiveInput()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Prescaler***
  Specifies the TI2 input prescaler value. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.
- ***uint32_t LL_TIM_ENCODER_InitTypeDef::IC2Filter***
  Specifies the TI2 input filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.

119.1.5 LL_TIM_HALLSENSOR_InitTypeDef

LL_TIM_HALLSENSOR_InitTypeDef is defined in the stm32u5xx_ll_tim.h

Data Fields

- ***uint32_t IC1Polarity***
- ***uint32_t IC1Prescaler***
- ***uint32_t IC1Filter***
- ***uint32_t CommutationDelay***

Field Documentation

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Polarity***
  Specifies the active edge of TI1 input. This parameter can be a value of *TIM_LL_EC_IC_POLARITY*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPolarity()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Prescaler***
  Specifies the TI1 input prescaler value. Prescaler must be set to get a maximum counter period longer than the time interval between 2 consecutive changes on the Hall inputs. This parameter can be a value of *TIM_LL_EC_ICPSC*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetPrescaler()**.

- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::IC1Filter***
  Specifies the TI1 input filter. This parameter can be a value of *TIM_LL_EC_IC_FILTER*.This feature can be modified afterwards using unitary function **LL_TIM_IC_SetFilter()**.
- ***uint32_t LL_TIM_HALLSENSOR_InitTypeDef::CommutationDelay***
  Specifies the compare value to be loaded into the Capture Compare Register. A positive pulse (TRGO event) is generated with a programmable delay every time a change occurs on the Hall inputs. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetCompareCH2()**.

119.1.6 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32u5xx_ll_tim.h

Data Fields

- ***uint32_t OSSRState***
- ***uint32_t OSSIState***
- ***uint32_t LockLevel***
- ***uint8_t DeadTime***
- ***uint16_t BreakState***
- ***uint32_t BreakPolarity***
- ***uint32_t BreakFilter***
- ***uint32_t BreakAFMode***
- ***uint32_t Break2State***
- ***uint32_t Break2Polarity***
- ***uint32_t Break2Filter***
- ***uint32_t Break2AFMode***
- ***uint32_t AutomaticOutput***

Field Documentation

- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSRState***
  Specifies the Off-State selection used in Run mode. This parameter can be a value of *TIM_LL_EC_OSSR*This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::OSSIState***
  Specifies the Off-State used in Idle state. This parameter can be a value of *TIM_LL_EC_OSSI*This feature can be modified afterwards using unitary function **LL_TIM_SetOffStates()**
  Note:
  – This bit-field cannot be modified as long as LOCK level 2 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::LockLevel***
  Specifies the LOCK level parameters. This parameter can be a value of *TIM_LL_EC_LOCKLEVEL*
  Note:
  – The LOCK bits can be written only once after the reset. Once the TIMx_BDTR register has been written, their content is frozen until the next reset.
- ***uint8_t LL_TIM_BDTR_InitTypeDef::DeadTime***
  Specifies the delay time between the switching-off and the switching-on of the outputs. This parameter can be a number between Min_Data = 0x00 and Max_Data = 0xFF.This feature can be modified afterwards using unitary function **LL_TIM_OC_SetDeadTime()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed.
- ***uint16_t LL_TIM_BDTR_InitTypeDef::BreakState***
  Specifies whether the TIM Break input is enabled or not. This parameter can be a value of *TIM_LL_EC_BREAK_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK()** or **LL_TIM_DisableBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakPolarity***
  Specifies the TIM Break Input pin polarity. This parameter can be a value of *TIM_LL_EC_BREAK_POLARITY*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakFilter***
  Specifies the TIM Break Filter. This parameter can be a value of *TIM_LL_EC_BREAK_FILTER*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::BreakAFMode***
  Specifies the alternate function mode of the break input. This parameter can be a value of *TIM_LL_EC_BREAK_AFMODE*This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2State***
  Specifies whether the TIM Break2 input is enabled or not. This parameter can be a value of *TIM_LL_EC_BREAK2_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableBRK2()** or **LL_TIM_DisableBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Polarity***
  Specifies the TIM Break2 Input pin polarity. This parameter can be a value of *TIM_LL_EC_BREAK2_POLARITY*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2Filter***
  Specifies the TIM Break2 Filter. This parameter can be a value of *TIM_LL_EC_BREAK2_FILTER*This feature can be modified afterwards using unitary function **LL_TIM_ConfigBRK2()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::Break2AFMode***
  Specifies the alternate function mode of the break2 input. This parameter can be a value of *TIM_LL_EC_BREAK2_AFMODE*This feature can be modified afterwards using unitary functions **LL_TIM_ConfigBRK2()**
  Note:
  – Bidirectional break input is only supported by advanced timers instances.
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.
- ***uint32_t LL_TIM_BDTR_InitTypeDef::AutomaticOutput***
  Specifies whether the TIM Automatic Output feature is enabled or not. This parameter can be a value of *TIM_LL_EC_AUTOMATICOUTPUT_ENABLE*This feature can be modified afterwards using unitary functions **LL_TIM_EnableAutomaticOutput()** or **LL_TIM_DisableAutomaticOutput()**
  Note:
  – This bit-field can not be modified as long as LOCK level 1 has been programmed.

119.2 TIM Firmware driver API description

The following section lists the various functions of the TIM library.

119.2.1 Detailed description of functions

LL_TIM_EnableCounter

Function name

__STATIC_INLINE void LL_TIM_EnableCounter (TIM_TypeDef * TIMx)

Function description

Enable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_EnableCounter

LL_TIM_DisableCounter

Function name

__STATIC_INLINE void LL_TIM_DisableCounter (TIM_TypeDef * TIMx)

Function description

Disable timer counter.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_DisableCounter

LL_TIM_IsEnabledCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledCounter (const TIM_TypeDef * TIMx)

Function description

Indicates whether the timer counter is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CEN LL_TIM_IsEnabledCounter

LL_TIM_EnableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_EnableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Enable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_EnableUpdateEvent

LL_TIM_DisableUpdateEvent

Function name

__STATIC_INLINE void LL_TIM_DisableUpdateEvent (TIM_TypeDef * TIMx)

Function description

Disable update event generation.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_DisableUpdateEvent

LL_TIM_IsEnabledUpdateEvent

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledUpdateEvent (const TIM_TypeDef * TIMx)

Function description

Indicates whether update event generation is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **Inverted:** state of bit (0 or 1).

Reference Manual to LL API cross reference:

- CR1 UDIS LL_TIM_IsEnabledUpdateEvent

LL_TIM_SetUpdateSource

Function name

__STATIC_INLINE void LL_TIM_SetUpdateSource (TIM_TypeDef * TIMx, uint32_t UpdateSource)

Function description

Set update event source.

Parameters

- **TIMx:** Timer instance
- **UpdateSource:** This parameter can be one of the following values:
  – LL_TIM_UPDATESOURCE_REGULAR
  – LL_TIM_UPDATESOURCE_COUNTER

Return values

- **None:**

Notes

- Update event source set to LL_TIM_UPDATESOURCE_REGULAR: any of the following events generate an update interrupt or DMA request if enabled: Counter overflow/underflowSetting the UG bitUpdate generation through the slave mode controller
- Update event source set to LL_TIM_UPDATESOURCE_COUNTER: only counter overflow/underflow generates an update interrupt or DMA request if enabled.

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_SetUpdateSource

LL_TIM_GetUpdateSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetUpdateSource (const TIM_TypeDef * TIMx)

Function description

Get actual event update source.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_UPDATESOURCE_REGULAR
  – LL_TIM_UPDATESOURCE_COUNTER

Reference Manual to LL API cross reference:

- CR1 URS LL_TIM_GetUpdateSource

LL_TIM_SetOnePulseMode

Function name

__STATIC_INLINE void LL_TIM_SetOnePulseMode (TIM_TypeDef * TIMx, uint32_t OnePulseMode)

Function description

Set one pulse mode (one shot v.s.

Parameters

- **TIMx:** Timer instance
- **OnePulseMode:** This parameter can be one of the following values:
  – LL_TIM_ONEPULSEMODE_SINGLE
  – LL_TIM_ONEPULSEMODE_REPETITIVE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_SetOnePulseMode

LL_TIM_GetOnePulseMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetOnePulseMode (const TIM_TypeDef * TIMx)

Function description

Get actual one pulse mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_ONEPULSEMODE_SINGLE
  – LL_TIM_ONEPULSEMODE_REPETITIVE

Reference Manual to LL API cross reference:

- CR1 OPM LL_TIM_GetOnePulseMode

LL_TIM_SetCounterMode

Function name

__STATIC_INLINE void LL_TIM_SetCounterMode (TIM_TypeDef * TIMx, uint32_t CounterMode)

Function description

Set the timer counter counting mode.

Parameters

- **TIMx:** Timer instance
- **CounterMode:** This parameter can be one of the following values:
  – LL_TIM_COUNTERMODE_UP
  – LL_TIM_COUNTERMODE_DOWN
  – LL_TIM_COUNTERMODE_CENTER_UP
  – LL_TIM_COUNTERMODE_CENTER_DOWN
  – LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.
- Switching from Center Aligned counter mode to Edge counter mode (or reverse) requires a timer reset to avoid unexpected direction due to DIR bit readonly in center aligned mode.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_SetCounterMode
- CR1 CMS LL_TIM_SetCounterMode

LL_TIM_GetCounterMode

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounterMode (const TIM_TypeDef * TIMx)

Function description

Get actual counter mode.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERMODE_UP
  - LL_TIM_COUNTERMODE_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP
  - LL_TIM_COUNTERMODE_CENTER_DOWN
  - LL_TIM_COUNTERMODE_CENTER_UP_DOWN

Notes

- Macro IS_TIM_COUNTER_MODE_SELECT_INSTANCE(TIMx) can be used to check whether or not the counter mode selection feature is supported by a timer instance.

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetCounterMode
- CR1 CMS LL_TIM_GetCounterMode

LL_TIM_EnableARRPreload

Function name

__STATIC_INLINE void LL_TIM_EnableARRPreload (TIM_TypeDef * TIMx)

Function description

Enable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_EnableARRPreload

LL_TIM_DisableARRPreload

Function name

__STATIC_INLINE void LL_TIM_DisableARRPreload (TIM_TypeDef * TIMx)

Function description

Disable auto-reload (ARR) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_DisableARRPreload

LL_TIM_IsEnabledARRPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledARRPreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether auto-reload (ARR) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 ARPE LL_TIM_IsEnabledARRPreload

LL_TIM_SetClockDivision

Function name

__STATIC_INLINE void LL_TIM_SetClockDivision (TIM_TypeDef * TIMx, uint32_t ClockDivision)

Function description

Set the division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance
- **ClockDivision:** This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_SetClockDivision

LL_TIM_GetClockDivision

Function name

__STATIC_INLINE uint32_t LL_TIM_GetClockDivision (const TIM_TypeDef * TIMx)

Function description

Get the actual division ratio between the timer clock and the sampling clock used by the dead-time generators (when supported) and the digital filters.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_CLOCKDIVISION_DIV1
  – LL_TIM_CLOCKDIVISION_DIV2
  – LL_TIM_CLOCKDIVISION_DIV4

Notes

- Macro IS_TIM_CLOCK_DIVISION_INSTANCE(TIMx) can be used to check whether or not the clock division feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- CR1 CKD LL_TIM_GetClockDivision

LL_TIM_SetCounter

Function name

__STATIC_INLINE void LL_TIM_SetCounter (TIM_TypeDef * TIMx, uint32_t Counter)

Function description

Set the counter value.

Parameters

- **TIMx:** Timer instance
- **Counter:** Counter value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Return values

- **None:**

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_SetCounter

LL_TIM_GetCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetCounter (const TIM_TypeDef * TIMx)

Function description

Get the counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Counter:** value (between Min_Data=0 and Max_Data=0xFFFF or 0xFFFFFFFF)

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the Counter value interpretation

Reference Manual to LL API cross reference:

- CNT CNT LL_TIM_GetCounter

LL_TIM_GetDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetDirection (const TIM_TypeDef * TIMx)

Function description

Get the current direction of the counter.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_COUNTERDIRECTION_UP
  - LL_TIM_COUNTERDIRECTION_DOWN

Reference Manual to LL API cross reference:

- CR1 DIR LL_TIM_GetDirection

LL_TIM_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Prescaler)

Function description

Set the prescaler value.

Parameters

- **TIMx:** Timer instance
- **Prescaler:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter clock frequency CK_CNT is equal to fCK_PSC / (PSC[15:0] + 1).
- The prescaler can be changed on the fly as this control register is buffered. The new prescaler ratio is taken into account at the next update event.
- Helper macro __LL_TIM_CALC_PSC can be used to calculate the Prescaler parameter

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_SetPrescaler

LL_TIM_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_GetPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the prescaler value.

Parameters

- **TIMx:** Timer instance

Return values

- **Prescaler:** value between Min_Data=0 and Max_Data=65535

Reference Manual to LL API cross reference:

- PSC PSC LL_TIM_GetPrescaler

LL_TIM_SetAutoReload

Function name

__STATIC_INLINE void LL_TIM_SetAutoReload (TIM_TypeDef * TIMx, uint32_t AutoReload)

Function description

Set the auto-reload value.

Parameters

- **TIMx:** Timer instance
- **AutoReload:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- The counter is blocked while the auto-reload value is null.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Helper macro __LL_TIM_CALC_ARR can be used to calculate the AutoReload parameter In case dithering is activated,macro __LL_TIM_CALC_ARR_DITHER can be used instead, to calculate the AutoReload parameter.

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_SetAutoReload

LL_TIM_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_TIM_GetAutoReload (const TIM_TypeDef * TIMx)

Function description

Get the auto-reload value.

Parameters

- **TIMx:** Timer instance

Return values

- **Auto-reload:** value

Notes

- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- If dithering is activated, pay attention to the returned value interpretation

Reference Manual to LL API cross reference:

- ARR ARR LL_TIM_GetAutoReload

LL_TIM_SetRepetitionCounter

Function name

__STATIC_INLINE void LL_TIM_SetRepetitionCounter (TIM_TypeDef * TIMx, uint32_t RepetitionCounter)

Function description

Set the repetition counter value.

Parameters

- **TIMx:** Timer instance
- **RepetitionCounter:** between Min_Data=0 and Max_Data=255 or 65535 for advanced timer.

Return values

- **None:**

Notes

- For advanced timer instances RepetitionCounter can be up to 65535.
- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_SetRepetitionCounter

LL_TIM_GetRepetitionCounter

Function name

__STATIC_INLINE uint32_t LL_TIM_GetRepetitionCounter (const TIM_TypeDef * TIMx)

Function description

Get the repetition counter value.

Parameters

- **TIMx:** Timer instance

Return values

- **Repetition:** counter value

Notes

- Macro IS_TIM_REPETITION_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a repetition counter.

Reference Manual to LL API cross reference:

- RCR REP LL_TIM_GetRepetitionCounter

LL_TIM_EnableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_EnableUIFRemap (TIM_TypeDef * TIMx)

Function description

Force a continuous copy of the update interrupt flag (UIF) into the timer counter register (bit 31).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- This allows both the counter value and a potential roll-over condition signalled by the UIFCPY flag to be read in an atomic way.

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_EnableUIFRemap

LL_TIM_DisableUIFRemap

Function name

__STATIC_INLINE void LL_TIM_DisableUIFRemap (TIM_TypeDef * TIMx)

Function description

Disable update interrupt flag (UIF) remapping.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UIFREMAP LL_TIM_DisableUIFRemap

LL_TIM_IsActiveUIFCPY

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveUIFCPY (const uint32_t Counter)

Function description

Indicate whether update interrupt flag (UIF) copy is set.

Parameters

- **Counter:** Counter value

Return values

- **State:** of bit (1 or 0).

LL_TIM_EnableDithering

Function name

__STATIC_INLINE void LL_TIM_EnableDithering (TIM_TypeDef * TIMx)

Function description

Enable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_EnableDithering

LL_TIM_DisableDithering

Function name

__STATIC_INLINE void LL_TIM_DisableDithering (TIM_TypeDef * TIMx)

Function description

Disable dithering.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_DisableDithering

LL_TIM_IsEnabledDithering

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDithering (const TIM_TypeDef * TIMx)

Function description

Indicates whether dithering is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DITHERING_INSTANCE(TIMx) can be used to check whether or not a timer instance provides dithering.

Reference Manual to LL API cross reference:

- CR1 DITHEN LL_TIM_IsEnabledDithering

LL_TIM_CC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_EnablePreload (TIM_TypeDef * TIMx)

Function description

Enable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- CCxE, CCxNE and OCxM bits are preloaded, after having been written, they are updated only when a commutation event (COM) occurs.
- Only on channels that have a complementary output.
- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_EnablePreload

LL_TIM_CC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_CC_DisablePreload (TIM_TypeDef * TIMx)

Function description

Disable the capture/compare control bits (CCxE, CCxNE and OCxM) preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_DisablePreload

LL_TIM_CC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledPreload (TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare control bits (CCxE, CCxNE and OCxM) preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 CCPC LL_TIM_CC_IsEnabledPreload

LL_TIM_CC_SetUpdate

Function name

__STATIC_INLINE void LL_TIM_CC_SetUpdate (TIM_TypeDef * TIMx, uint32_t CCUpdateSource)

Function description

Set the updated source of the capture/compare control bits (CCxE, CCxNE and OCxM).

Parameters

- **TIMx:** Timer instance
- **CCUpdateSource:** This parameter can be one of the following values:
  - LL_TIM_CCUPDATESOURCE_COMG_ONLY
  - LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Return values

- **None:**

Notes

- Macro IS_TIM_COMMUTATION_EVENT_INSTANCE(TIMx) can be used to check whether or not a timer instance is able to generate a commutation event.

Reference Manual to LL API cross reference:

- CR2 CCUS LL_TIM_CC_SetUpdate

LL_TIM_CC_SetDMAReqTrigger

Function name

__STATIC_INLINE void LL_TIM_CC_SetDMAReqTrigger (TIM_TypeDef * TIMx, uint32_t DMAReqTrigger)

Function description

Set the trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance
- **DMAReqTrigger:** This parameter can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_SetDMAReqTrigger

LL_TIM_CC_GetDMAReqTrigger

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_GetDMAReqTrigger (const TIM_TypeDef * TIMx)

Function description

Get actual trigger of the capture/compare DMA request.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_CCDMAREQUEST_CC
  - LL_TIM_CCDMAREQUEST_UPDATE

Reference Manual to LL API cross reference:

- CR2 CCDS LL_TIM_CC_GetDMAReqTrigger

LL_TIM_CC_SetLockLevel

Function name

__STATIC_INLINE void LL_TIM_CC_SetLockLevel (TIM_TypeDef * TIMx, uint32_t LockLevel)

Function description

Set the lock level to freeze the configuration of several capture/compare parameters.

Parameters

- **TIMx:** Timer instance
- **LockLevel:** This parameter can be one of the following values:
  – LL_TIM_LOCKLEVEL_OFF
  – LL_TIM_LOCKLEVEL_1
  – LL_TIM_LOCKLEVEL_2
  – LL_TIM_LOCKLEVEL_3

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not the lock mechanism is supported by a timer instance.

Reference Manual to LL API cross reference:

- BDTR LOCK LL_TIM_CC_SetLockLevel

LL_TIM_CC_EnableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_EnableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Enable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH1N
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH2N
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH3N
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH4N
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_EnableChannel
- CCER CC1NE LL_TIM_CC_EnableChannel
- CCER CC2E LL_TIM_CC_EnableChannel
- CCER CC2NE LL_TIM_CC_EnableChannel
- CCER CC3E LL_TIM_CC_EnableChannel
- CCER CC3NE LL_TIM_CC_EnableChannel
- CCER CC4E LL_TIM_CC_EnableChannel
- CCER CC4NE LL_TIM_CC_EnableChannel
- CCER CC5E LL_TIM_CC_EnableChannel
- CCER CC6E LL_TIM_CC_EnableChannel

LL_TIM_CC_DisableChannel

Function name

__STATIC_INLINE void LL_TIM_CC_DisableChannel (TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Disable capture/compare channels.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_DisableChannel
- CCER CC1NE LL_TIM_CC_DisableChannel
- CCER CC2E LL_TIM_CC_DisableChannel
- CCER CC2NE LL_TIM_CC_DisableChannel
- CCER CC3E LL_TIM_CC_DisableChannel
- CCER CC3NE LL_TIM_CC_DisableChannel
- CCER CC4E LL_TIM_CC_DisableChannel
- CCER CC4NE LL_TIM_CC_DisableChannel
- CCER CC5E LL_TIM_CC_DisableChannel
- CCER CC6E LL_TIM_CC_DisableChannel

LL_TIM_CC_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_TIM_CC_IsEnabledChannel (const TIM_TypeDef * TIMx, uint32_t Channels)

Function description

Indicate whether channel(s) is(are) enabled.

Parameters

- **TIMx:** Timer instance
- **Channels:** This parameter can be a combination of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH1N
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH2N
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH3N
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH4N
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCER CC1E LL_TIM_CC_IsEnabledChannel
- CCER CC1NE LL_TIM_CC_IsEnabledChannel
- CCER CC2E LL_TIM_CC_IsEnabledChannel
- CCER CC2NE LL_TIM_CC_IsEnabledChannel
- CCER CC3E LL_TIM_CC_IsEnabledChannel
- CCER CC3NE LL_TIM_CC_IsEnabledChannel
- CCER CC4E LL_TIM_CC_IsEnabledChannel
- CCER CC4NE LL_TIM_CC_IsEnabledChannel
- CCER CC5E LL_TIM_CC_IsEnabledChannel
- CCER CC6E LL_TIM_CC_IsEnabledChannel

LL_TIM_OC_ConfigOutput

Function name

__STATIC_INLINE void LL_TIM_OC_ConfigOutput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_OCPOLARITY_HIGH or LL_TIM_OCPOLARITY_LOW
  - LL_TIM_OCIDLESTATE_LOW or LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_OC_ConfigOutput
- CCMR1 CC2S LL_TIM_OC_ConfigOutput
- CCMR2 CC3S LL_TIM_OC_ConfigOutput
- CCMR2 CC4S LL_TIM_OC_ConfigOutput
- CCMR3 CC5S LL_TIM_OC_ConfigOutput
- CCMR3 CC6S LL_TIM_OC_ConfigOutput
- CCER CC1P LL_TIM_OC_ConfigOutput
- CCER CC2P LL_TIM_OC_ConfigOutput
- CCER CC3P LL_TIM_OC_ConfigOutput
- CCER CC4P LL_TIM_OC_ConfigOutput
- CCER CC5P LL_TIM_OC_ConfigOutput
- CCER CC6P LL_TIM_OC_ConfigOutput
- CR2 OIS1 LL_TIM_OC_ConfigOutput
- CR2 OIS2 LL_TIM_OC_ConfigOutput
- CR2 OIS3 LL_TIM_OC_ConfigOutput
- CR2 OIS4 LL_TIM_OC_ConfigOutput
- CR2 OIS5 LL_TIM_OC_ConfigOutput
- CR2 OIS6 LL_TIM_OC_ConfigOutput

LL_TIM_OC_SetMode

Function name

__STATIC_INLINE void LL_TIM_OC_SetMode (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Mode)

Function description

Define the behavior of the output reference signal OCxREF from which OCx and OCxN (when relevant) are derived.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **Mode:** This parameter can be one of the following values:
  - LL_TIM_OCMODE_FROZEN
  - LL_TIM_OCMODE_ACTIVE
  - LL_TIM_OCMODE_INACTIVE
  - LL_TIM_OCMODE_TOGGLE
  - LL_TIM_OCMODE_FORCED_INACTIVE
  - LL_TIM_OCMODE_FORCED_ACTIVE
  - LL_TIM_OCMODE_PWM1
  - LL_TIM_OCMODE_PWM2
  - LL_TIM_OCMODE_RETRIG_OPM1
  - LL_TIM_OCMODE_RETRIG_OPM2
  - LL_TIM_OCMODE_COMBINED_PWM1
  - LL_TIM_OCMODE_COMBINED_PWM2
  - LL_TIM_OCMODE_ASYMMETRIC_PWM1
  - LL_TIM_OCMODE_ASYMMETRIC_PWM2
  - LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  - LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_SetMode
- CCMR1 OC2M LL_TIM_OC_SetMode
- CCMR2 OC3M LL_TIM_OC_SetMode
- CCMR2 OC4M LL_TIM_OC_SetMode
- CCMR3 OC5M LL_TIM_OC_SetMode
- CCMR3 OC6M LL_TIM_OC_SetMode

LL_TIM_OC_GetMode

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetMode (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the output compare mode of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_OCMODE_FROZEN
  – LL_TIM_OCMODE_ACTIVE
  – LL_TIM_OCMODE_INACTIVE
  – LL_TIM_OCMODE_TOGGLE
  – LL_TIM_OCMODE_FORCED_INACTIVE
  – LL_TIM_OCMODE_FORCED_ACTIVE
  – LL_TIM_OCMODE_PWM1
  – LL_TIM_OCMODE_PWM2
  – LL_TIM_OCMODE_RETRIG_OPM1
  – LL_TIM_OCMODE_RETRIG_OPM2
  – LL_TIM_OCMODE_COMBINED_PWM1
  – LL_TIM_OCMODE_COMBINED_PWM2
  – LL_TIM_OCMODE_ASYMMETRIC_PWM1
  – LL_TIM_OCMODE_ASYMMETRIC_PWM2
  – LL_TIM_OCMODE_PULSE_ON_COMPARE (for channel 3 or channel 4 only)
  – LL_TIM_OCMODE_DIRECTION_OUTPUT (for channel 3 or channel 4 only)

Reference Manual to LL API cross reference:

- CCMR1 OC1M LL_TIM_OC_GetMode
- CCMR1 OC2M LL_TIM_OC_GetMode
- CCMR2 OC3M LL_TIM_OC_GetMode
- CCMR2 OC4M LL_TIM_OC_GetMode
- CCMR3 OC5M LL_TIM_OC_GetMode
- CCMR3 OC6M LL_TIM_OC_GetMode

LL_TIM_OC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_OC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH1N
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH2N
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH3N
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH4N
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6
- **Polarity:** This parameter can be one of the following values:
  – LL_TIM_OCPOLARITY_HIGH
  – LL_TIM_OCPOLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_SetPolarity
- CCER CC1NP LL_TIM_OC_SetPolarity
- CCER CC2P LL_TIM_OC_SetPolarity
- CCER CC2NP LL_TIM_OC_SetPolarity
- CCER CC3P LL_TIM_OC_SetPolarity
- CCER CC3NP LL_TIM_OC_SetPolarity
- CCER CC4P LL_TIM_OC_SetPolarity
- CCER CC4NP LL_TIM_OC_SetPolarity
- CCER CC5P LL_TIM_OC_SetPolarity
- CCER CC6P LL_TIM_OC_SetPolarity

LL_TIM_OC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the polarity of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH1N
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH2N
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH3N
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH4N
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_OCPOLARITY_HIGH
  - LL_TIM_OCPOLARITY_LOW

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_OC_GetPolarity
- CCER CC1NP LL_TIM_OC_GetPolarity
- CCER CC2P LL_TIM_OC_GetPolarity
- CCER CC2NP LL_TIM_OC_GetPolarity
- CCER CC3P LL_TIM_OC_GetPolarity
- CCER CC3NP LL_TIM_OC_GetPolarity
- CCER CC4P LL_TIM_OC_GetPolarity
- CCER CC4NP LL_TIM_OC_GetPolarity
- CCER CC5P LL_TIM_OC_GetPolarity
- CCER CC6P LL_TIM_OC_GetPolarity

LL_TIM_OC_SetIdleState

Function name

__STATIC_INLINE void LL_TIM_OC_SetIdleState (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t IdleState)

Function description

Set the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH1N
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH2N
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH3N
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH4N
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6
- **IdleState:** This parameter can be one of the following values:
  – LL_TIM_OCIDLESTATE_LOW
  – LL_TIM_OCIDLESTATE_HIGH

Return values

- **None:**

Notes

- This function is significant only for the timer instances supporting the break feature. Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS2 LL_TIM_OC_SetIdleState
- CR2 OIS2N LL_TIM_OC_SetIdleState
- CR2 OIS3 LL_TIM_OC_SetIdleState
- CR2 OIS3N LL_TIM_OC_SetIdleState
- CR2 OIS4 LL_TIM_OC_SetIdleState
- CR2 OIS4N LL_TIM_OC_SetIdleState
- CR2 OIS5 LL_TIM_OC_SetIdleState
- CR2 OIS6 LL_TIM_OC_SetIdleState

LL_TIM_OC_GetIdleState

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetIdleState (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the IDLE state of an output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH1N
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH2N
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH3N
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH4N
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_OCIDLESTATE_LOW
  – LL_TIM_OCIDLESTATE_HIGH

Reference Manual to LL API cross reference:

- CR2 OIS1 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS2 LL_TIM_OC_GetIdleState
- CR2 OIS2N LL_TIM_OC_GetIdleState
- CR2 OIS3 LL_TIM_OC_GetIdleState
- CR2 OIS3N LL_TIM_OC_GetIdleState
- CR2 OIS4 LL_TIM_OC_GetIdleState
- CR2 OIS4N LL_TIM_OC_GetIdleState
- CR2 OIS5 LL_TIM_OC_GetIdleState
- CR2 OIS6 LL_TIM_OC_GetIdleState

LL_TIM_OC_EnableFast

Function name

__STATIC_INLINE void LL_TIM_OC_EnableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Acts only if the channel is configured in PWM1 or PWM2 mode.

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_EnableFast
- CCMR1 OC2FE LL_TIM_OC_EnableFast
- CCMR2 OC3FE LL_TIM_OC_EnableFast
- CCMR2 OC4FE LL_TIM_OC_EnableFast
- CCMR3 OC5FE LL_TIM_OC_EnableFast
- CCMR3 OC6FE LL_TIM_OC_EnableFast

LL_TIM_OC_DisableFast

Function name

__STATIC_INLINE void LL_TIM_OC_DisableFast (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable fast mode for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_DisableFast
- CCMR1 OC2FE LL_TIM_OC_DisableFast
- CCMR2 OC3FE LL_TIM_OC_DisableFast
- CCMR2 OC4FE LL_TIM_OC_DisableFast
- CCMR3 OC5FE LL_TIM_OC_DisableFast
- CCMR3 OC6FE LL_TIM_OC_DisableFast

LL_TIM_OC_IsEnabledFast

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledFast (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether fast mode is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1FE LL_TIM_OC_IsEnabledFast
- CCMR1 OC2FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC3FE LL_TIM_OC_IsEnabledFast
- CCMR2 OC4FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC5FE LL_TIM_OC_IsEnabledFast
- CCMR3 OC6FE LL_TIM_OC_IsEnabledFast

LL_TIM_OC_EnablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_EnablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
  – LL_TIM_CHANNEL_CH5
  – LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_EnablePreload
- CCMR1 OC2PE LL_TIM_OC_EnablePreload
- CCMR2 OC3PE LL_TIM_OC_EnablePreload
- CCMR2 OC4PE LL_TIM_OC_EnablePreload
- CCMR3 OC5PE LL_TIM_OC_EnablePreload
- CCMR3 OC6PE LL_TIM_OC_EnablePreload

LL_TIM_OC_DisablePreload

Function name

__STATIC_INLINE void LL_TIM_OC_DisablePreload (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable compare register (TIMx_CCRx) preload for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_DisablePreload
- CCMR1 OC2PE LL_TIM_OC_DisablePreload
- CCMR2 OC3PE LL_TIM_OC_DisablePreload
- CCMR2 OC4PE LL_TIM_OC_DisablePreload
- CCMR3 OC5PE LL_TIM_OC_DisablePreload
- CCMR3 OC6PE LL_TIM_OC_DisablePreload

LL_TIM_OC_IsEnabledPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledPreload (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates whether compare register (TIMx_CCRx) preload is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CCMR1 OC1PE LL_TIM_OC_IsEnabledPreload
- CCMR1 OC2PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC3PE LL_TIM_OC_IsEnabledPreload
- CCMR2 OC4PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC5PE LL_TIM_OC_IsEnabledPreload
- CCMR3 OC6PE LL_TIM_OC_IsEnabledPreload

LL_TIM_OC_EnableClear

Function name

__STATIC_INLINE void LL_TIM_OC_EnableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Enable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_EnableClear
- CCMR1 OC2CE LL_TIM_OC_EnableClear
- CCMR2 OC3CE LL_TIM_OC_EnableClear
- CCMR2 OC4CE LL_TIM_OC_EnableClear
- CCMR3 OC5CE LL_TIM_OC_EnableClear
- CCMR3 OC6CE LL_TIM_OC_EnableClear

LL_TIM_OC_DisableClear

Function name

__STATIC_INLINE void LL_TIM_OC_DisableClear (TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Disable clearing the output channel on an external event.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **None:**

Notes

- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_DisableClear
- CCMR1 OC2CE LL_TIM_OC_DisableClear
- CCMR2 OC3CE LL_TIM_OC_DisableClear
- CCMR2 OC4CE LL_TIM_OC_DisableClear
- CCMR3 OC5CE LL_TIM_OC_DisableClear
- CCMR3 OC6CE LL_TIM_OC_DisableClear

LL_TIM_OC_IsEnabledClear

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_IsEnabledClear (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Indicates clearing the output channel on an external event is enabled for the output channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6

Return values

- **State:** of bit (1 or 0).

Notes

- This function enables clearing the output channel on an external event.
- This function can only be used in Output compare and PWM modes. It does not work in Forced mode.
- Macro IS_TIM_OCXREF_CLEAR_INSTANCE(TIMx) can be used to check whether or not a timer instance can clear the OCxREF signal on an external event.

Reference Manual to LL API cross reference:

- CCMR1 OC1CE LL_TIM_OC_IsEnabledClear
- CCMR1 OC2CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC3CE LL_TIM_OC_IsEnabledClear
- CCMR2 OC4CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC5CE LL_TIM_OC_IsEnabledClear
- CCMR3 OC6CE LL_TIM_OC_IsEnabledClear

LL_TIM_OC_SetDeadTime

Function name

__STATIC_INLINE void LL_TIM_OC_SetDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the dead-time delay (delay inserted between the rising edge of the OCxREF signal and the rising edge of the Ocx and OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter

Reference Manual to LL API cross reference:

- BDTR DTG LL_TIM_OC_SetDeadTime

LL_TIM_OC_SetCompareCH1

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH1 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 1 (TIMx_CCR1).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_SetCompareCH1

LL_TIM_OC_SetCompareCH2

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH2 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 2 (TIMx_CCR2).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_SetCompareCH2

LL_TIM_OC_SetCompareCH3

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH3 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 3 (TIMx_CCR3).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_SetCompareCH3

LL_TIM_OC_SetCompareCH4

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH4 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 4 (TIMx_CCR4).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- In 32-bit timer implementations compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_SetCompareCH4

LL_TIM_OC_SetCompareCH5

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH5 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 5 (TIMx_CCR5).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_SetCompareCH5

LL_TIM_OC_SetCompareCH6

Function name

__STATIC_INLINE void LL_TIM_OC_SetCompareCH6 (TIM_TypeDef * TIMx, uint32_t CompareValue)

Function description

Set compare value for output channel 6 (TIMx_CCR6).

Parameters

- **TIMx:** Timer instance
- **CompareValue:** between Min_Data=0 and Max_Data=65535

Return values

- **None:**

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, CompareValue can be calculated with macro __LL_TIM_CALC_DELAY_DITHER .

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_SetCompareCH6

LL_TIM_OC_GetCompareCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH1 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR1) set for output channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not output channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_OC_GetCompareCH1

LL_TIM_OC_GetCompareCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH2 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR2) set for output channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not output channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_OC_GetCompareCH2

LL_TIM_OC_GetCompareCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH3 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR3) set for output channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not output channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_OC_GetCompareCH3

LL_TIM_OC_GetCompareCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH4 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR4) set for output channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned compare value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not output channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_OC_GetCompareCH4

LL_TIM_OC_GetCompareCH5

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH5 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR5) set for output channel 5.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC5_INSTANCE(TIMx) can be used to check whether or not output channel 5 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR5 CCR5 LL_TIM_OC_GetCompareCH5

LL_TIM_OC_GetCompareCH6

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetCompareCH6 (const TIM_TypeDef * TIMx)

Function description

Get compare value (TIMx_CCR6) set for output channel 6.

Parameters

- **TIMx:** Timer instance

Return values

- **CompareValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- Macro IS_TIM_CC6_INSTANCE(TIMx) can be used to check whether or not output channel 6 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR6 CCR6 LL_TIM_OC_GetCompareCH6

LL_TIM_SetCH5CombinedChannels

Function name

__STATIC_INLINE void LL_TIM_SetCH5CombinedChannels (TIM_TypeDef * TIMx, uint32_t GroupCH5)

Function description

Select on which reference signal the OC5REF is combined to.

Parameters

- **TIMx:** Timer instance
- **GroupCH5:** This parameter can be a combination of the following values:
  – LL_TIM_GROUPCH5_NONE
  – LL_TIM_GROUPCH5_OC1REFC
  – LL_TIM_GROUPCH5_OC2REFC
  – LL_TIM_GROUPCH5_OC3REFC

Return values

- **None:**

Notes

- Macro IS_TIM_COMBINED3PHASEPWM_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the combined 3-phase PWM mode.

Reference Manual to LL API cross reference:

- CCR5 GC5C3 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C2 LL_TIM_SetCH5CombinedChannels
- CCR5 GC5C1 LL_TIM_SetCH5CombinedChannels

LL_TIM_OC_SetPulseWidthPrescaler

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidthPrescaler (TIM_TypeDef * TIMx, uint32_t PulseWidthPrescaler)

Function description

Set the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance
- **PulseWidthPrescaler:** This parameter can be one of the following values:
  – LL_TIM_PWPRSC_X1
  – LL_TIM_PWPRSC_X2
  – LL_TIM_PWPRSC_X4
  – LL_TIM_PWPRSC_X8
  – LL_TIM_PWPRSC_X16
  – LL_TIM_PWPRSC_X32
  – LL_TIM_PWPRSC_X64
  – LL_TIM_PWPRSC_X128

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_SetPulseWidthPrescaler

LL_TIM_OC_GetPulseWidthPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidthPrescaler (const TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width prescaler.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_PWPRSC_X1
  – LL_TIM_PWPRSC_X2
  – LL_TIM_PWPRSC_X4
  – LL_TIM_PWPRSC_X8
  – LL_TIM_PWPRSC_X16
  – LL_TIM_PWPRSC_X32
  – LL_TIM_PWPRSC_X64
  – LL_TIM_PWPRSC_X128

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PWPRSC LL_TIM_OC_GetPulseWidthPrescaler

LL_TIM_OC_SetPulseWidth

Function name

__STATIC_INLINE void LL_TIM_OC_SetPulseWidth (TIM_TypeDef * TIMx, uint32_t PulseWidth)

Function description

Set the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance
- **PulseWidth:** This parameter can be between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_SetPulseWidth

LL_TIM_OC_GetPulseWidth

Function name

__STATIC_INLINE uint32_t LL_TIM_OC_GetPulseWidth (const TIM_TypeDef * TIMx)

Function description

Get the pulse on compare pulse width duration.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_PULSEONCOMPARE_INSTANCE(TIMx) can be used to check whether or not the pulse on compare feature is supported by the timer instance.

Reference Manual to LL API cross reference:

- ECR PW LL_TIM_OC_GetPulseWidth

LL_TIM_IC_Config

Function name

__STATIC_INLINE void LL_TIM_IC_Config (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t Configuration)

Function description

Configure input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI or LL_TIM_ACTIVEINPUT_INDIRECTTI or LL_TIM_ACTIVEINPUT_TRC
  - LL_TIM_ICPSC_DIV1 or ... or LL_TIM_ICPSC_DIV8
  - LL_TIM_IC_FILTER_FDIV1 or ... or LL_TIM_IC_FILTER_FDIV32_N8
  - LL_TIM_IC_POLARITY_RISING or LL_TIM_IC_POLARITY_FALLING or LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_Config
- CCMR1 IC1PSC LL_TIM_IC_Config
- CCMR1 IC1F LL_TIM_IC_Config
- CCMR1 CC2S LL_TIM_IC_Config
- CCMR1 IC2PSC LL_TIM_IC_Config
- CCMR1 IC2F LL_TIM_IC_Config
- CCMR2 CC3S LL_TIM_IC_Config
- CCMR2 IC3PSC LL_TIM_IC_Config
- CCMR2 IC3F LL_TIM_IC_Config
- CCMR2 CC4S LL_TIM_IC_Config
- CCMR2 IC4PSC LL_TIM_IC_Config
- CCMR2 IC4F LL_TIM_IC_Config
- CCER CC1P LL_TIM_IC_Config
- CCER CC1NP LL_TIM_IC_Config
- CCER CC2P LL_TIM_IC_Config
- CCER CC2NP LL_TIM_IC_Config
- CCER CC3P LL_TIM_IC_Config
- CCER CC3NP LL_TIM_IC_Config
- CCER CC4P LL_TIM_IC_Config
- CCER CC4NP LL_TIM_IC_Config

LL_TIM_IC_SetActiveInput

Function name

__STATIC_INLINE void LL_TIM_IC_SetActiveInput (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICActiveInput)

Function description

Set the active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
- **ICActiveInput:** This parameter can be one of the following values:
  – LL_TIM_ACTIVEINPUT_DIRECTTI
  – LL_TIM_ACTIVEINPUT_INDIRECTTI
  – LL_TIM_ACTIVEINPUT_TRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_SetActiveInput
- CCMR1 CC2S LL_TIM_IC_SetActiveInput
- CCMR2 CC3S LL_TIM_IC_SetActiveInput
- CCMR2 CC4S LL_TIM_IC_SetActiveInput

LL_TIM_IC_GetActiveInput

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetActiveInput (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current active input.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_ACTIVEINPUT_DIRECTTI
  - LL_TIM_ACTIVEINPUT_INDIRECTTI
  - LL_TIM_ACTIVEINPUT_TRC

Reference Manual to LL API cross reference:

- CCMR1 CC1S LL_TIM_IC_GetActiveInput
- CCMR1 CC2S LL_TIM_IC_GetActiveInput
- CCMR2 CC3S LL_TIM_IC_GetActiveInput
- CCMR2 CC4S LL_TIM_IC_GetActiveInput

LL_TIM_IC_SetPrescaler

Function name

__STATIC_INLINE void LL_TIM_IC_SetPrescaler (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPrescaler)

Function description

Set the prescaler of input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPrescaler:** This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_SetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_SetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_SetPrescaler

LL_TIM_IC_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPrescaler (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current prescaler value acting on an input channel.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_ICPSC_DIV1
  – LL_TIM_ICPSC_DIV2
  – LL_TIM_ICPSC_DIV4
  – LL_TIM_ICPSC_DIV8

Reference Manual to LL API cross reference:

- CCMR1 IC1PSC LL_TIM_IC_GetPrescaler
- CCMR1 IC2PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC3PSC LL_TIM_IC_GetPrescaler
- CCMR2 IC4PSC LL_TIM_IC_GetPrescaler

LL_TIM_IC_SetFilter

Function name

__STATIC_INLINE void LL_TIM_IC_SetFilter (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICFilter)

Function description

Set the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
- **ICFilter:** This parameter can be one of the following values:
  – LL_TIM_IC_FILTER_FDIV1
  – LL_TIM_IC_FILTER_FDIV1_N2
  – LL_TIM_IC_FILTER_FDIV1_N4
  – LL_TIM_IC_FILTER_FDIV1_N8
  – LL_TIM_IC_FILTER_FDIV2_N6
  – LL_TIM_IC_FILTER_FDIV2_N8
  – LL_TIM_IC_FILTER_FDIV4_N6
  – LL_TIM_IC_FILTER_FDIV4_N8
  – LL_TIM_IC_FILTER_FDIV8_N6
  – LL_TIM_IC_FILTER_FDIV8_N8
  – LL_TIM_IC_FILTER_FDIV16_N5
  – LL_TIM_IC_FILTER_FDIV16_N6
  – LL_TIM_IC_FILTER_FDIV16_N8
  – LL_TIM_IC_FILTER_FDIV32_N5
  – LL_TIM_IC_FILTER_FDIV32_N6
  – LL_TIM_IC_FILTER_FDIV32_N8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_SetFilter
- CCMR1 IC2F LL_TIM_IC_SetFilter
- CCMR2 IC3F LL_TIM_IC_SetFilter
- CCMR2 IC4F LL_TIM_IC_SetFilter

LL_TIM_IC_GetFilter

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetFilter (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the input filter duration.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_IC_FILTER_FDIV1
  - LL_TIM_IC_FILTER_FDIV1_N2
  - LL_TIM_IC_FILTER_FDIV1_N4
  - LL_TIM_IC_FILTER_FDIV1_N8
  - LL_TIM_IC_FILTER_FDIV2_N6
  - LL_TIM_IC_FILTER_FDIV2_N8
  - LL_TIM_IC_FILTER_FDIV4_N6
  - LL_TIM_IC_FILTER_FDIV4_N8
  - LL_TIM_IC_FILTER_FDIV8_N6
  - LL_TIM_IC_FILTER_FDIV8_N8
  - LL_TIM_IC_FILTER_FDIV16_N5
  - LL_TIM_IC_FILTER_FDIV16_N6
  - LL_TIM_IC_FILTER_FDIV16_N8
  - LL_TIM_IC_FILTER_FDIV32_N5
  - LL_TIM_IC_FILTER_FDIV32_N6
  - LL_TIM_IC_FILTER_FDIV32_N8

Reference Manual to LL API cross reference:

- CCMR1 IC1F LL_TIM_IC_GetFilter
- CCMR1 IC2F LL_TIM_IC_GetFilter
- CCMR2 IC3F LL_TIM_IC_GetFilter
- CCMR2 IC4F LL_TIM_IC_GetFilter

LL_TIM_IC_SetPolarity

Function name

__STATIC_INLINE void LL_TIM_IC_SetPolarity (TIM_TypeDef * TIMx, uint32_t Channel, uint32_t ICPolarity)

Function description

Set the input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
- **ICPolarity:** This parameter can be one of the following values:
  - LL_TIM_IC_POLARITY_RISING
  - LL_TIM_IC_POLARITY_FALLING
  - LL_TIM_IC_POLARITY_BOTHEDGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_SetPolarity
- CCER CC1NP LL_TIM_IC_SetPolarity
- CCER CC2P LL_TIM_IC_SetPolarity
- CCER CC2NP LL_TIM_IC_SetPolarity
- CCER CC3P LL_TIM_IC_SetPolarity
- CCER CC3NP LL_TIM_IC_SetPolarity
- CCER CC4P LL_TIM_IC_SetPolarity
- CCER CC4NP LL_TIM_IC_SetPolarity

LL_TIM_IC_GetPolarity

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetPolarity (const TIM_TypeDef * TIMx, uint32_t Channel)

Function description

Get the current input channel polarity.

Parameters

- **TIMx:** Timer instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_IC_POLARITY_RISING
  – LL_TIM_IC_POLARITY_FALLING
  – LL_TIM_IC_POLARITY_BOTHEDGE

Reference Manual to LL API cross reference:

- CCER CC1P LL_TIM_IC_GetPolarity
- CCER CC1NP LL_TIM_IC_GetPolarity
- CCER CC2P LL_TIM_IC_GetPolarity
- CCER CC2NP LL_TIM_IC_GetPolarity
- CCER CC3P LL_TIM_IC_GetPolarity
- CCER CC3NP LL_TIM_IC_GetPolarity
- CCER CC4P LL_TIM_IC_GetPolarity
- CCER CC4NP LL_TIM_IC_GetPolarity

LL_TIM_IC_EnableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_EnableXORCombination (TIM_TypeDef * TIMx)

Function description

Connect the TIMx_CH1, CH2 and CH3 pins to the TI1 input (XOR combination).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_EnableXORCombination

LL_TIM_IC_DisableXORCombination

Function name

__STATIC_INLINE void LL_TIM_IC_DisableXORCombination (TIM_TypeDef * TIMx)

Function description

Disconnect the TIMx_CH1, CH2 and CH3 pins from the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_DisableXORCombination

LL_TIM_IC_IsEnabledXORCombination

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_IsEnabledXORCombination (const TIM_TypeDef * TIMx)

Function description

Indicates whether the TIMx_CH1, CH2 and CH3 pins are connectected to the TI1 input.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_XOR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an XOR input.

Reference Manual to LL API cross reference:

- CR2 TI1S LL_TIM_IC_IsEnabledXORCombination

LL_TIM_IC_GetCaptureCH1

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH1 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 1.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC1_INSTANCE(TIMx) can be used to check whether or not input channel 1 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR1 CCR1 LL_TIM_IC_GetCaptureCH1

LL_TIM_IC_GetCaptureCH2

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH2 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 2.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC2_INSTANCE(TIMx) can be used to check whether or not input channel 2 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR2 CCR2 LL_TIM_IC_GetCaptureCH2

LL_TIM_IC_GetCaptureCH3

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH3 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 3.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC3_INSTANCE(TIMx) can be used to check whether or not input channel 3 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR3 CCR3 LL_TIM_IC_GetCaptureCH3

LL_TIM_IC_GetCaptureCH4

Function name

__STATIC_INLINE uint32_t LL_TIM_IC_GetCaptureCH4 (const TIM_TypeDef * TIMx)

Function description

Get captured value for input channel 4.

Parameters

- **TIMx:** Timer instance

Return values

- **CapturedValue:** (between Min_Data=0 and Max_Data=65535)

Notes

- In 32-bit timer implementations returned captured value can be between 0x00000000 and 0xFFFFFFFF.
- Macro IS_TIM_32B_COUNTER_INSTANCE(TIMx) can be used to check whether or not a timer instance supports a 32 bits counter.
- Macro IS_TIM_CC4_INSTANCE(TIMx) can be used to check whether or not input channel 4 is supported by a timer instance.
- If dithering is activated, pay attention to the returned value interpretation.

Reference Manual to LL API cross reference:

- CCR4 CCR4 LL_TIM_IC_GetCaptureCH4

LL_TIM_EnableExternalClock

Function name

__STATIC_INLINE void LL_TIM_EnableExternalClock (TIM_TypeDef * TIMx)

Function description

Enable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- When external clock mode 2 is enabled the counter is clocked by any active edge on the ETRF signal.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_EnableExternalClock

LL_TIM_DisableExternalClock

Function name

__STATIC_INLINE void LL_TIM_DisableExternalClock (TIM_TypeDef * TIMx)

Function description

Disable external clock mode 2.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_DisableExternalClock

LL_TIM_IsEnabledExternalClock

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledExternalClock (const TIM_TypeDef * TIMx)

Function description

Indicate whether external clock mode 2 is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR ECE LL_TIM_IsEnabledExternalClock

LL_TIM_SetClockSource

Function name

__STATIC_INLINE void LL_TIM_SetClockSource (TIM_TypeDef * TIMx, uint32_t ClockSource)

Function description

Set the clock source of the counter clock.

Parameters

- **TIMx:** Timer instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_TIM_CLOCKSOURCE_INTERNAL
  - LL_TIM_CLOCKSOURCE_EXT_MODE1
  - LL_TIM_CLOCKSOURCE_EXT_MODE2

Return values

- **None:**

Notes

- when selected clock source is external clock mode 1, the timer input the external clock is applied is selected by calling the LL_TIM_SetTriggerInput() function. This timer input must be configured by calling the LL_TIM_IC_Config() function.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE1_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode1.
- Macro IS_TIM_CLOCKSOURCE_ETRMODE2_INSTANCE(TIMx) can be used to check whether or not a timer instance supports external clock mode2.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetClockSource
- SMCR ECE LL_TIM_SetClockSource

LL_TIM_SetEncoderMode

Function name

__STATIC_INLINE void LL_TIM_SetEncoderMode (TIM_TypeDef * TIMx, uint32_t EncoderMode)

Function description

Set the encoder interface mode.

Parameters

- **TIMx:** Timer instance
- **EncoderMode:** This parameter can be one of the following values:
  - LL_TIM_ENCODERMODE_X2_TI1
  - LL_TIM_ENCODERMODE_X2_TI2
  - LL_TIM_ENCODERMODE_X4_TI12
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2
  - LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2
  - LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12
  - LL_TIM_ENCODERMODE_X1_TI1
  - LL_TIM_ENCODERMODE_X1_TI2

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_INTERFACE_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the encoder mode.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetEncoderMode

LL_TIM_SetTriggerOutput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput (TIM_TypeDef * TIMx, uint32_t TimerSynchronization)

Function description

Set the trigger output (TRGO) used for timer synchronization .

Parameters

- **TIMx:** Timer instance
- **TimerSynchronization:** This parameter can be one of the following values:
  - LL_TIM_TRGO_RESET
  - LL_TIM_TRGO_ENABLE
  - LL_TIM_TRGO_UPDATE
  - LL_TIM_TRGO_CC1IF
  - LL_TIM_TRGO_OC1REF
  - LL_TIM_TRGO_OC2REF
  - LL_TIM_TRGO_OC3REF
  - LL_TIM_TRGO_OC4REF
  - LL_TIM_TRGO_ENCODERCLK

Return values

- **None:**

Notes

- Macro IS_TIM_MASTER_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a master timer.

Reference Manual to LL API cross reference:

- CR2 MMS LL_TIM_SetTriggerOutput

LL_TIM_SetTriggerOutput2

Function name

__STATIC_INLINE void LL_TIM_SetTriggerOutput2 (TIM_TypeDef * TIMx, uint32_t ADCSynchronization)

Function description

Set the trigger output 2 (TRGO2) used for ADC synchronization .

Parameters

- **TIMx:** Timer Instance
- **ADCSynchronization:** This parameter can be one of the following values:
  – LL_TIM_TRGO2_RESET
  – LL_TIM_TRGO2_ENABLE
  – LL_TIM_TRGO2_UPDATE
  – LL_TIM_TRGO2_CC1F
  – LL_TIM_TRGO2_OC1
  – LL_TIM_TRGO2_OC2
  – LL_TIM_TRGO2_OC3
  – LL_TIM_TRGO2_OC4
  – LL_TIM_TRGO2_OC5
  – LL_TIM_TRGO2_OC6
  – LL_TIM_TRGO2_OC4_RISINGFALLING
  – LL_TIM_TRGO2_OC6_RISINGFALLING
  – LL_TIM_TRGO2_OC4_RISING_OC6_RISING
  – LL_TIM_TRGO2_OC4_RISING_OC6_FALLING
  – LL_TIM_TRGO2_OC5_RISING_OC6_RISING
  – LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

Return values

- **None:**

Notes

- Macro IS_TIM_TRGO2_INSTANCE(TIMx) can be used to check whether or not a timer instance can be used for ADC synchronization.

Reference Manual to LL API cross reference:

- CR2 MMS2 LL_TIM_SetTriggerOutput2

LL_TIM_SetSlaveMode

Function name

__STATIC_INLINE void LL_TIM_SetSlaveMode (TIM_TypeDef * TIMx, uint32_t SlaveMode)

Function description

Set the synchronization mode of a slave timer.

Parameters

- **TIMx:** Timer instance
- **SlaveMode:** This parameter can be one of the following values:
  – LL_TIM_SLAVEMODE_DISABLED
  – LL_TIM_SLAVEMODE_RESET
  – LL_TIM_SLAVEMODE_GATED
  – LL_TIM_SLAVEMODE_TRIGGER
  – LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER
  – LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR SMS LL_TIM_SetSlaveMode

LL_TIM_SetTriggerInput

Function name

__STATIC_INLINE void LL_TIM_SetTriggerInput (TIM_TypeDef * TIMx, uint32_t TriggerInput)

Function description

Set the selects the trigger input to be used to synchronize the counter.

Parameters

- **TIMx:** Timer instance
- **TriggerInput:** This parameter can be one of the following values:
  - LL_TIM_TS_ITR0
  - LL_TIM_TS_ITR1
  - LL_TIM_TS_ITR2
  - LL_TIM_TS_ITR3
  - LL_TIM_TS_ITR4
  - LL_TIM_TS_ITR5
  - LL_TIM_TS_ITR6
  - LL_TIM_TS_ITR7
  - LL_TIM_TS_ITR8
  - LL_TIM_TS_ITR11
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_EnableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_EnableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Enable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_EnableMasterSlaveMode

LL_TIM_DisableMasterSlaveMode

Function name

__STATIC_INLINE void LL_TIM_DisableMasterSlaveMode (TIM_TypeDef * TIMx)

Function description

Disable the Master/Slave mode.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_DisableMasterSlaveMode

LL_TIM_IsEnabledMasterSlaveMode

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledMasterSlaveMode (const TIM_TypeDef * TIMx)

Function description

Indicates whether the Master/Slave mode is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR MSM LL_TIM_IsEnabledMasterSlaveMode

LL_TIM_ConfigETR

Function name

__STATIC_INLINE void LL_TIM_ConfigETR (TIM_TypeDef * TIMx, uint32_t ETRPolarity, uint32_t ETRPrescaler, uint32_t ETRFilter)

Function description

Configure the external trigger (ETR) input.

Parameters

- **TIMx:** Timer instance
- **ETRPolarity:** This parameter can be one of the following values:
  – LL_TIM_ETR_POLARITY_NONINVERTED
  – LL_TIM_ETR_POLARITY_INVERTED
- **ETRPrescaler:** This parameter can be one of the following values:
  – LL_TIM_ETR_PRESCALER_DIV1
  – LL_TIM_ETR_PRESCALER_DIV2
  – LL_TIM_ETR_PRESCALER_DIV4
  – LL_TIM_ETR_PRESCALER_DIV8
- **ETRFilter:** This parameter can be one of the following values:
  – LL_TIM_ETR_FILTER_FDIV1
  – LL_TIM_ETR_FILTER_FDIV1_N2
  – LL_TIM_ETR_FILTER_FDIV1_N4
  – LL_TIM_ETR_FILTER_FDIV1_N8
  – LL_TIM_ETR_FILTER_FDIV2_N6
  – LL_TIM_ETR_FILTER_FDIV2_N8
  – LL_TIM_ETR_FILTER_FDIV4_N6
  – LL_TIM_ETR_FILTER_FDIV4_N8
  – LL_TIM_ETR_FILTER_FDIV8_N6
  – LL_TIM_ETR_FILTER_FDIV8_N8
  – LL_TIM_ETR_FILTER_FDIV16_N5
  – LL_TIM_ETR_FILTER_FDIV16_N6
  – LL_TIM_ETR_FILTER_FDIV16_N8
  – LL_TIM_ETR_FILTER_FDIV32_N5
  – LL_TIM_ETR_FILTER_FDIV32_N6
  – LL_TIM_ETR_FILTER_FDIV32_N8

Return values

- **None:**

Notes

- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an external trigger input.

Reference Manual to LL API cross reference:

- SMCR ETP LL_TIM_ConfigETR
- SMCR ETPS LL_TIM_ConfigETR
- SMCR ETF LL_TIM_ConfigETR

LL_TIM_SetETRSource

Function name

__STATIC_INLINE void LL_TIM_SetETRSource (TIM_TypeDef * TIMx, uint32_t ETRSource)

Function description

Select the external trigger (ETR) input source.

Parameters

- **TIMx:** Timer instance
- **ETRSource:** This parameter can be one of the following values:

Return values

- **None:**

Notes

- Macro IS_TIM_ETR_INSTANCE(TIMx) can be used to check whether or not a timer instance supports ETR source selection.

Reference Manual to LL API cross reference:

- AF1 ETRSEL LL_TIM_SetETRSource

LL_TIM_EnableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_EnableSMSPreload (TIM_TypeDef * TIMx)

Function description

Enable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_EnableSMSPreload

LL_TIM_DisableSMSPreload

Function name

__STATIC_INLINE void LL_TIM_DisableSMSPreload (TIM_TypeDef * TIMx)

Function description

Disable SMS preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_DisableSMSPreload

LL_TIM_IsEnabledSMSPreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledSMSPreload (const TIM_TypeDef * TIMx)

Function description

Indicate whether SMS preload is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPE LL_TIM_IsEnabledSMSPreload

LL_TIM_SetSMSPreloadSource

Function name

__STATIC_INLINE void LL_TIM_SetSMSPreloadSource (TIM_TypeDef * TIMx, uint32_t PreloadSource)

Function description

Set the preload source of SMS.

Parameters

- **TIMx:** Timer instance
- **PreloadSource:** This parameter can be one of the following values:
  – LL_TIM_SMSPS_TIMUPDATE
  – LL_TIM_SMSPS_INDEX

Return values

- **None:**

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_SetSMSPreloadSource
-

LL_TIM_GetSMSPreloadSource

Function name

__STATIC_INLINE uint32_t LL_TIM_GetSMSPreloadSource (const TIM_TypeDef * TIMx)

Function description

Get the preload source of SMS.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_SMSPS_TIMUPDATE
  – LL_TIM_SMSPS_INDEX

Notes

- Macro IS_TIM_SMS_PRELOAD_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the preload of SMS field in SMCR register.

Reference Manual to LL API cross reference:

- SMCR SMSPS LL_TIM_GetSMSPreloadSource
-

LL_TIM_EnableBRK

Function name

__STATIC_INLINE void LL_TIM_EnableBRK (TIM_TypeDef * TIMx)

Function description

Enable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_EnableBRK

LL_TIM_DisableBRK

Function name

__STATIC_INLINE void LL_TIM_DisableBRK (TIM_TypeDef * TIMx)

Function description

Disable the break function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR BKE LL_TIM_DisableBRK

LL_TIM_ConfigBRK

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK (TIM_TypeDef * TIMx, uint32_t BreakPolarity, uint32_t BreakFilter, uint32_t BreakAFMode)

Function description

Configure the break input.

Parameters

- **TIMx:** Timer instance
- **BreakPolarity:** This parameter can be one of the following values:
  – LL_TIM_BREAK_POLARITY_LOW
  – LL_TIM_BREAK_POLARITY_HIGH
- **BreakFilter:** This parameter can be one of the following values:
  – LL_TIM_BREAK_FILTER_FDIV1
  – LL_TIM_BREAK_FILTER_FDIV1_N2
  – LL_TIM_BREAK_FILTER_FDIV1_N4
  – LL_TIM_BREAK_FILTER_FDIV1_N8
  – LL_TIM_BREAK_FILTER_FDIV2_N6
  – LL_TIM_BREAK_FILTER_FDIV2_N8
  – LL_TIM_BREAK_FILTER_FDIV4_N6
  – LL_TIM_BREAK_FILTER_FDIV4_N8
  – LL_TIM_BREAK_FILTER_FDIV8_N6
  – LL_TIM_BREAK_FILTER_FDIV8_N8
  – LL_TIM_BREAK_FILTER_FDIV16_N5
  – LL_TIM_BREAK_FILTER_FDIV16_N6
  – LL_TIM_BREAK_FILTER_FDIV16_N8
  – LL_TIM_BREAK_FILTER_FDIV32_N5
  – LL_TIM_BREAK_FILTER_FDIV32_N6
  – LL_TIM_BREAK_FILTER_FDIV32_N8
- **BreakAFMode:** This parameter can be one of the following values:
  – LL_TIM_BREAK_AFMODE_INPUT
  – LL_TIM_BREAK_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BKBID bit set), the Break input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, BreakAFMode must be set to LL_TIM_BREAK_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BKP LL_TIM_ConfigBRK
- BDTR BKF LL_TIM_ConfigBRK
- BDTR BKBID LL_TIM_ConfigBRK

LL_TIM_DisarmBRK

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK (TIM_TypeDef * TIMx)

Function description

Disarm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output .

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_DisarmBRK

LL_TIM_EnableBRK2

Function name

__STATIC_INLINE void LL_TIM_EnableBRK2 (TIM_TypeDef * TIMx)

Function description

Enable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_EnableBRK2

LL_TIM_DisableBRK2

Function name

__STATIC_INLINE void LL_TIM_DisableBRK2 (TIM_TypeDef * TIMx)

Function description

Disable the break 2 function.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

Reference Manual to LL API cross reference:

- BDTR BK2E LL_TIM_DisableBRK2

LL_TIM_ConfigBRK2

Function name

__STATIC_INLINE void LL_TIM_ConfigBRK2 (TIM_TypeDef * TIMx, uint32_t Break2Polarity, uint32_t Break2Filter, uint32_t Break2AFMode)

Function description

Configure the break 2 input.

Parameters

- **TIMx:** Timer instance
- **Break2Polarity:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_POLARITY_LOW
  - LL_TIM_BREAK2_POLARITY_HIGH
- **Break2Filter:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_FILTER_FDIV1
  - LL_TIM_BREAK2_FILTER_FDIV1_N2
  - LL_TIM_BREAK2_FILTER_FDIV1_N4
  - LL_TIM_BREAK2_FILTER_FDIV1_N8
  - LL_TIM_BREAK2_FILTER_FDIV2_N6
  - LL_TIM_BREAK2_FILTER_FDIV2_N8
  - LL_TIM_BREAK2_FILTER_FDIV4_N6
  - LL_TIM_BREAK2_FILTER_FDIV4_N8
  - LL_TIM_BREAK2_FILTER_FDIV8_N6
  - LL_TIM_BREAK2_FILTER_FDIV8_N8
  - LL_TIM_BREAK2_FILTER_FDIV16_N5
  - LL_TIM_BREAK2_FILTER_FDIV16_N6
  - LL_TIM_BREAK2_FILTER_FDIV16_N8
  - LL_TIM_BREAK2_FILTER_FDIV32_N5
  - LL_TIM_BREAK2_FILTER_FDIV32_N6
  - LL_TIM_BREAK2_FILTER_FDIV32_N8
- **Break2AFMode:** This parameter can be one of the following values:
  - LL_TIM_BREAK2_AFMODE_INPUT
  - LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Return values

- **None:**

Notes

- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.
- Bidirectional mode is only supported by advanced timer instances. Macro IS_TIM_ADVANCED_INSTANCE(TIMx) can be used to check whether or not a timer instance is an advanced-control timer.
- In bidirectional mode (BK2BID bit set), the Break 2 input is configured both in input mode and in open drain output mode. Any active Break event will assert a low logic level on the Break 2 input to indicate an internal break event to external devices.
- When bidirectional mode isn't supported, Break2AFMode must be set to LL_TIM_BREAK2_AFMODE_INPUT.

Reference Manual to LL API cross reference:

- BDTR BK2P LL_TIM_ConfigBRK2
- BDTR BK2F LL_TIM_ConfigBRK2
- BDTR BK2BID LL_TIM_ConfigBRK2

LL_TIM_DisarmBRK2

Function name

__STATIC_INLINE void LL_TIM_DisarmBRK2 (TIM_TypeDef * TIMx)

Function description

Disarm the break 2 input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The break 2 input can be disarmed only when it is configured in bidirectional mode and when when MOE is reset.
- Purpose is to be able to have the input voltage back to high-state, whatever the time constant on the output.

Reference Manual to LL API cross reference:

- BDTR BK2DSRM LL_TIM_DisarmBRK2

LL_TIM_SetOffStates

Function name

__STATIC_INLINE void LL_TIM_SetOffStates (TIM_TypeDef * TIMx, uint32_t OffStateIdle, uint32_t OffStateRun)

Function description

Select the outputs off state (enabled v.s.

Parameters

- **TIMx:** Timer instance
- **OffStateIdle:** This parameter can be one of the following values:
  – LL_TIM_OSSI_DISABLE
  – LL_TIM_OSSI_ENABLE
- **OffStateRun:** This parameter can be one of the following values:
  – LL_TIM_OSSR_DISABLE
  – LL_TIM_OSSR_ENABLE

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR OSSI LL_TIM_SetOffStates
- BDTR OSSR LL_TIM_SetOffStates

LL_TIM_EnableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_EnableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Enable automatic output (MOE can be set by software or automatically when a break input is active).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_EnableAutomaticOutput

LL_TIM_DisableAutomaticOutput

Function name

__STATIC_INLINE void LL_TIM_DisableAutomaticOutput (TIM_TypeDef * TIMx)

Function description

Disable automatic output (MOE can be set only by software).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_DisableAutomaticOutput

LL_TIM_IsEnabledAutomaticOutput

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAutomaticOutput (const TIM_TypeDef * TIMx)

Function description

Indicate whether automatic output is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR AOE LL_TIM_IsEnabledAutomaticOutput

LL_TIM_EnableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_EnableAllOutputs (TIM_TypeDef * TIMx)

Function description

Enable the outputs (set the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_EnableAllOutputs

LL_TIM_DisableAllOutputs

Function name

__STATIC_INLINE void LL_TIM_DisableAllOutputs (TIM_TypeDef * TIMx)

Function description

Disable the outputs (reset the MOE bit in TIMx_BDTR register).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The MOE bit in TIMx_BDTR register allows to enable /disable the outputs by software and is reset in case of break or break2 event.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_DisableAllOutputs

LL_TIM_IsEnabledAllOutputs

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAllOutputs (const TIM_TypeDef * TIMx)

Function description

Indicates whether outputs are enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.

Reference Manual to LL API cross reference:

- BDTR MOE LL_TIM_IsEnabledAllOutputs

LL_TIM_EnableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_EnableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Enable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  – LL_TIM_BREAK_INPUT_BKIN
  – LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  – LL_TIM_BKIN_SOURCE_BKIN
  – LL_TIM_BKIN_SOURCE_BKCOMP1
  – LL_TIM_BKIN_SOURCE_BKCOMP2 (*)
  – LL_TIM_BKIN_SOURCE_MDF1

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_EnableBreakInputSource
- AF1 BKCMP1E LL_TIM_EnableBreakInputSource
- AF1 BKCMP2E LL_TIM_EnableBreakInputSource
- AF1 BKDF1BK0E LL_TIM_EnableBreakInputSource
- AF2 BK2INE LL_TIM_EnableBreakInputSource
- AF2 BK2CMP1E LL_TIM_EnableBreakInputSource
- AF2 BK2CMP2E LL_TIM_EnableBreakInputSource
- AF2 BK2DF1BK1E LL_TIM_EnableBreakInputSource

LL_TIM_DisableBreakInputSource

Function name

__STATIC_INLINE void LL_TIM_DisableBreakInputSource (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source)

Function description

Disable the signals connected to the designated timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  – LL_TIM_BREAK_INPUT_BKIN
  – LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  – LL_TIM_BKIN_SOURCE_BKIN
  – LL_TIM_BKIN_SOURCE_BKCOMP1
  – LL_TIM_BKIN_SOURCE_BKCOMP2 (*)
  – LL_TIM_BKIN_SOURCE_MDF1

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINE LL_TIM_DisableBreakInputSource
- AF1 BKCMP1E LL_TIM_DisableBreakInputSource
- AF1 BKCMP2E LL_TIM_DisableBreakInputSource
- AF1 BKDF1BK0E LL_TIM_DisableBreakInputSource
- AF2 BK2INE LL_TIM_DisableBreakInputSource
- AF2 BK2CMP1E LL_TIM_DisableBreakInputSource
- AF2 BK2CMP2E LL_TIM_DisableBreakInputSource
- AF2 BK2DF1BK1E LL_TIM_DisableBreakInputSource

LL_TIM_SetBreakInputSourcePolarity

Function name

__STATIC_INLINE void LL_TIM_SetBreakInputSourcePolarity (TIM_TypeDef * TIMx, uint32_t BreakInput, uint32_t Source, uint32_t Polarity)

Function description

Set the polarity of the break signal for the timer break input.

Parameters

- **TIMx:** Timer instance
- **BreakInput:** This parameter can be one of the following values:
  – LL_TIM_BREAK_INPUT_BKIN
  – LL_TIM_BREAK_INPUT_BKIN2
- **Source:** This parameter can be one of the following values:
  – LL_TIM_BKIN_SOURCE_BKIN
  – LL_TIM_BKIN_SOURCE_BKCOMP1
  – LL_TIM_BKIN_SOURCE_BKCOMP2 (*)
- **Polarity:** This parameter can be one of the following values:
  – LL_TIM_BKIN_POLARITY_LOW
  – LL_TIM_BKIN_POLARITY_HIGH

  (*) Value not defined in all devices.

Return values

- **None:**

Notes

- Macro IS_TIM_BREAKSOURCE_INSTANCE(TIMx) can be used to check whether or not a timer instance allows for break input selection.

Reference Manual to LL API cross reference:

- AF1 BKINP LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP1P LL_TIM_SetBreakInputSourcePolarity
- AF1 BKCMP2P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2INP LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP1P LL_TIM_SetBreakInputSourcePolarity
- AF2 BK2CMP2P LL_TIM_SetBreakInputSourcePolarity

LL_TIM_EnableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_EnableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Enable asymmetrical deadtime.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_EnableAsymmetricalDeadTime

LL_TIM_DisableAsymmetricalDeadTime

Function name

__STATIC_INLINE void LL_TIM_DisableAsymmetricalDeadTime (TIM_TypeDef * TIMx)

Function description

Disable asymmetrical dead-time.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_DisableAsymmetricalDeadTime

LL_TIM_IsEnabledAsymmetricalDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledAsymmetricalDeadTime (const TIM_TypeDef * TIMx)

Function description

Indicates whether asymmetrical deadtime is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not a timer instance provides asymmetrical deadtime.

Reference Manual to LL API cross reference:

- DTR2 DTAE LL_TIM_IsEnabledAsymmetricalDeadTime

LL_TIM_SetFallingDeadTime

Function name

__STATIC_INLINE void LL_TIM_SetFallingDeadTime (TIM_TypeDef * TIMx, uint32_t DeadTime)

Function description

Set the falling edge dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance
- **DeadTime:** between Min_Data=0 and Max_Data=255

Return values

- **None:**

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- Helper macro __LL_TIM_CALC_DEADTIME can be used to calculate the DeadTime parameter
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_SetFallingDeadTime

LL_TIM_GetFallingDeadTime

Function name

__STATIC_INLINE uint32_t LL_TIM_GetFallingDeadTime (const TIM_TypeDef * TIMx)

Function description

Get the falling edge dead-time delay (delay inserted between the falling edge of the OCxREF signal and the rising edge of OCxN signals).

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be between Min_Data=0 and Max_Data=255:

Notes

- Macro IS_TIM_DEADTIME_ASYMMETRICAL_INSTANCE(TIMx) can be used to check whether or not asymmetrical dead-time insertion feature is supported by a timer instance.
- This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

Reference Manual to LL API cross reference:

- DTR2 DTGF LL_TIM_GetFallingDeadTime

LL_TIM_EnableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_EnableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Enable deadtime preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_EnableDeadTimePreload

LL_TIM_DisableDeadTimePreload

Function name

__STATIC_INLINE void LL_TIM_DisableDeadTimePreload (TIM_TypeDef * TIMx)

Function description

Disable dead-time preload.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_DisableDeadTimePreload

LL_TIM_IsEnabledDeadTimePreload

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDeadTimePreload (const TIM_TypeDef * TIMx)

Function description

Indicates whether deadtime preload is activated.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides deadtime preload.

Reference Manual to LL API cross reference:

- DTR2 DTPE LL_TIM_IsEnabledDeadTimePreload

LL_TIM_ConfigDMABurst

Function name

__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength, uint32_t DMABurstSource)

Function description

Configures the timer DMA burst feature.

Parameters

- **TIMx:** Timer instance
- **DMABurstBaseAddress:** This parameter can be one of the following values:
  - LL_TIM_DMABURST_BASEADDR_CR1
  - LL_TIM_DMABURST_BASEADDR_CR2
  - LL_TIM_DMABURST_BASEADDR_SMCR
  - LL_TIM_DMABURST_BASEADDR_DIER
  - LL_TIM_DMABURST_BASEADDR_SR
  - LL_TIM_DMABURST_BASEADDR_EGR
  - LL_TIM_DMABURST_BASEADDR_CCMR1
  - LL_TIM_DMABURST_BASEADDR_CCMR2
  - LL_TIM_DMABURST_BASEADDR_CCER
  - LL_TIM_DMABURST_BASEADDR_CNT
  - LL_TIM_DMABURST_BASEADDR_PSC
  - LL_TIM_DMABURST_BASEADDR_ARR
  - LL_TIM_DMABURST_BASEADDR_RCR
  - LL_TIM_DMABURST_BASEADDR_CCR1
  - LL_TIM_DMABURST_BASEADDR_CCR2
  - LL_TIM_DMABURST_BASEADDR_CCR3
  - LL_TIM_DMABURST_BASEADDR_CCR4
  - LL_TIM_DMABURST_BASEADDR_BDTR
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_CCMR3
  - LL_TIM_DMABURST_BASEADDR_DTR2
  - LL_TIM_DMABURST_BASEADDR_ECR
  - LL_TIM_DMABURST_BASEADDR_TISEL
  - LL_TIM_DMABURST_BASEADDR_AF1
  - LL_TIM_DMABURST_BASEADDR_AF2
  - LL_TIM_DMABURST_BASEADDR_OR1

- **DMABurstLength:** This parameter can be one of the following values:
  – LL_TIM_DMABURST_LENGTH_1TRANSFER
  – LL_TIM_DMABURST_LENGTH_2TRANSFERS
  – LL_TIM_DMABURST_LENGTH_3TRANSFERS
  – LL_TIM_DMABURST_LENGTH_4TRANSFERS
  – LL_TIM_DMABURST_LENGTH_5TRANSFERS
  – LL_TIM_DMABURST_LENGTH_6TRANSFERS
  – LL_TIM_DMABURST_LENGTH_7TRANSFERS
  – LL_TIM_DMABURST_LENGTH_8TRANSFERS
  – LL_TIM_DMABURST_LENGTH_9TRANSFERS
  – LL_TIM_DMABURST_LENGTH_10TRANSFERS
  – LL_TIM_DMABURST_LENGTH_11TRANSFERS
  – LL_TIM_DMABURST_LENGTH_12TRANSFERS
  – LL_TIM_DMABURST_LENGTH_13TRANSFERS
  – LL_TIM_DMABURST_LENGTH_14TRANSFERS
  – LL_TIM_DMABURST_LENGTH_15TRANSFERS
  – LL_TIM_DMABURST_LENGTH_16TRANSFERS
  – LL_TIM_DMABURST_LENGTH_17TRANSFERS
  – LL_TIM_DMABURST_LENGTH_18TRANSFERS
  – LL_TIM_DMABURST_LENGTH_19TRANSFERS
  – LL_TIM_DMABURST_LENGTH_20TRANSFERS
  – LL_TIM_DMABURST_LENGTH_21TRANSFERS
  – LL_TIM_DMABURST_LENGTH_22TRANSFERS
  – LL_TIM_DMABURST_LENGTH_23TRANSFERS
  – LL_TIM_DMABURST_LENGTH_24TRANSFERS
  – LL_TIM_DMABURST_LENGTH_25TRANSFERS
  – LL_TIM_DMABURST_LENGTH_26TRANSFERS
- **DMABurstSource:** This parameter can be one of the following values:
  – LL_TIM_DMA_UPDATE
  – LL_TIM_DMA_CC1
  – LL_TIM_DMA_CC2
  – LL_TIM_DMA_CC3
  – LL_TIM_DMA_CC4
  – LL_TIM_DMA_COM
  – LL_TIM_DMA_TRIGGER

Return values

- **None:**

Notes

- Macro IS_TIM_DMABURST_INSTANCE(TIMx) can be used to check whether or not a timer instance supports the DMA burst mode.

Reference Manual to LL API cross reference:

- DCR DBL LL_TIM_ConfigDMABurst
- DCR DBA LL_TIM_ConfigDMABurst

LL_TIM_EnableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_EnableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Enable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_EnableEncoderIndex

LL_TIM_DisableEncoderIndex

Function name

__STATIC_INLINE void LL_TIM_DisableEncoderIndex (TIM_TypeDef * TIMx)

Function description

Disable encoder index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_DisableEncoderIndex

LL_TIM_IsEnabledEncoderIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledEncoderIndex (const TIM_TypeDef * TIMx)

Function description

Indicate whether encoder index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IE LL_TIM_IsEnabledEncoderIndex

LL_TIM_SetIndexDirection

Function name

__STATIC_INLINE void LL_TIM_SetIndexDirection (TIM_TypeDef * TIMx, uint32_t IndexDirection)

Function description

Set index direction.

Parameters

- **TIMx:** Timer instance
- **IndexDirection:** This parameter can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_SetIndexDirection

LL_TIM_GetIndexDirection

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexDirection (const TIM_TypeDef * TIMx)

Function description

Get actual index direction.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_UP
  - LL_TIM_INDEX_DOWN

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_GetIndexDirection

LL_TIM_SetIndexblanking

Function name

__STATIC_INLINE void LL_TIM_SetIndexblanking (TIM_TypeDef * TIMx, uint32_t Indexblanking)

Function description

Set index blanking.

Parameters

- **TIMx:** Timer instance
- **Indexblanking:** This parameter can be one of the following values:
  - LL_TIM_INDEX_BLANK_ALWAYS
  - LL_TIM_INDEX_BLANK_TI3
  - LL_TIM_INDEX_BLANK_TI4

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IBLK LL_TIM_SetIndexblanking

LL_TIM_GetIndexblanking

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexblanking (const TIM_TypeDef * TIMx)

Function description

Get actual index blanking.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  - LL_TIM_INDEX_BLANK_ALWAYS
  - LL_TIM_INDEX_BLANK_TI3
  - LL_TIM_INDEX_BLANK_TI4

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IBLK LL_TIM_GetIndexblanking

LL_TIM_EnableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_EnableFirstIndex (TIM_TypeDef * TIMx)

Function description

Enable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_EnableFirstIndex

LL_TIM_DisableFirstIndex

Function name

__STATIC_INLINE void LL_TIM_DisableFirstIndex (TIM_TypeDef * TIMx)

Function description

Disable first index.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_DisableFirstIndex

LL_TIM_IsEnabledFirstIndex

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledFirstIndex (const TIM_TypeDef * TIMx)

Function description

Indicates whether first index is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR FIDX LL_TIM_IsEnabledFirstIndex

LL_TIM_SetIndexPositionning

Function name

__STATIC_INLINE void LL_TIM_SetIndexPositionning (TIM_TypeDef * TIMx, uint32_t IndexPositionning)

Function description

Set index positioning.

Parameters

- **TIMx:** Timer instance
- **IndexPositionning:** This parameter can be one of the following values:
  – LL_TIM_INDEX_POSITION_DOWN_DOWN
  – LL_TIM_INDEX_POSITION_DOWN_UP
  – LL_TIM_INDEX_POSITION_UP_DOWN
  – LL_TIM_INDEX_POSITION_UP_UP
  – LL_TIM_INDEX_POSITION_DOWN
  – LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_SetIndexPositionning

LL_TIM_GetIndexPositionning

Function name

__STATIC_INLINE uint32_t LL_TIM_GetIndexPositionning (const TIM_TypeDef * TIMx)

Function description

Get actual index positioning.

Parameters

- **TIMx:** Timer instance

Return values

- **Returned:** value can be one of the following values:
  – LL_TIM_INDEX_POSITION_DOWN_DOWN
  – LL_TIM_INDEX_POSITION_DOWN_UP
  – LL_TIM_INDEX_POSITION_UP_DOWN
  – LL_TIM_INDEX_POSITION_UP_UP
  – LL_TIM_INDEX_POSITION_DOWN
  – LL_TIM_INDEX_POSITION_UP

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IPOS LL_TIM_GetIndexPositionning

LL_TIM_ConfigIDX

Function name

__STATIC_INLINE void LL_TIM_ConfigIDX (TIM_TypeDef * TIMx, uint32_t Configuration)

Function description

Configure encoder index.

Parameters

- **TIMx:** Timer instance
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_TIM_INDEX_UP or LL_TIM_INDEX_DOWN or LL_TIM_INDEX_UP_DOWN
  - LL_TIM_INDEX_BLANK_ALWAYS or LL_TIM_INDEX_BLANK_TI3 or LL_TIM_INDEX_BLANK_TI4
  - LL_TIM_INDEX_ALL or LL_TIM_INDEX_FIRST_ONLY
  - LL_TIM_INDEX_POSITION_DOWN_DOWN or ... or LL_TIM_INDEX_POSITION_UP

Return values

- **None:**

Notes

- Macro IS_TIM_INDEX_INSTANCE(TIMx) can be used to check whether or not a timer instance provides an index input.

Reference Manual to LL API cross reference:

- ECR IDIR LL_TIM_ConfigIDX
- ECR IBLK LL_TIM_ConfigIDX
- ECR FIDX LL_TIM_ConfigIDX
- ECR IPOS LL_TIM_ConfigIDX

LL_TIM_SetRemap

Function name

__STATIC_INLINE void LL_TIM_SetRemap (TIM_TypeDef * TIMx, uint32_t Remap)

Function description

Remap TIM inputs (input channel, internal/external triggers).

Parameters

- **TIMx:** Timer instance
- **Remap:** Remap param depends on the TIMx. Description available only in CHM version of the User Manual (not in .pdf). Otherwise see Reference Manual description of TISEL registers.

Return values

- **None:**

Notes

- Macro IS_TIM_REMAP_INSTANCE(TIMx) can be used to check whether or not a some timer inputs can be remapped.

Reference Manual to LL API cross reference:

- TIM1_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI2SEL LL_TIM_SetRemap
- TIM2_TISEL TI4SEL LL_TIM_SetRemap
- TIM3_TISEL TI1SEL LL_TIM_SetRemap
- TIM3_TISEL TI2SEL LL_TIM_SetRemap
- TIM4_TISEL TI1SEL LL_TIM_SetRemap
- TIM4_TISEL TI2SEL LL_TIM_SetRemap
- TIM5_TISEL TI1SEL LL_TIM_SetRemap
- TIM5_TISEL TI2SEL LL_TIM_SetRemap
- TIM8_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI1SEL LL_TIM_SetRemap
- TIM15_TISEL TI2SEL LL_TIM_SetRemap
- TIM16_TISEL TI1SEL LL_TIM_SetRemap
- TIM17_TISEL TI1SEL LL_TIM_SetRemap
-

LL_TIM_EnableHSE32

Function name

__STATIC_INLINE void LL_TIM_EnableHSE32 (TIM_TypeDef * TIMx)

Function description

Enable request for HSE/32 clock used for TISEL remap.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Only TIM16 and TIM17 support HSE/32 remap
- The Cut1.x contains a limitation when using HSE/32 as input capture for TIM16
- Bug ID 56: On TIM16, the HSE/32 input capture requires the set of HSE32EN bit of TIM17 Option Register

Reference Manual to LL API cross reference:

- OR1 HSE32EN LL_TIM_EnableHSE32

LL_TIM_DisableHSE32

Function name

__STATIC_INLINE void LL_TIM_DisableHSE32 (TIM_TypeDef * TIMx)

Function description

Disable request for HSE/32 clock used for TISEL remap.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Only TIM16 and TIM17 support HSE/32 remap

Reference Manual to LL API cross reference:

- OR1 HSE32EN LL_TIM_DisableHSE32

LL_TIM_IsEnabledHSE32

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledHSE32 (const TIM_TypeDef * TIMx)

Function description

Indicate whether request for HSE/32 clock is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Only TIM16 and TIM17 support HSE/32 remap

Reference Manual to LL API cross reference:

- OR1 HSE32EN LL_TIM_IsEnabledHSE32

LL_TIM_SetOCRefClearInputSource

Function name

__STATIC_INLINE void LL_TIM_SetOCRefClearInputSource (TIM_TypeDef * TIMx, uint32_t OCRefClearInputSource)

Function description

Set the OCREF clear input source.

Parameters

- **TIMx:** Timer instance
- **OCRefClearInputSource:** This parameter can be one of the following values:
  - LL_TIM_OCREF_CLR_INT_ETR
  - LL_TIM_OCREF_CLR_INT_COMP1
  - LL_TIM_OCREF_CLR_INT_COMP2 (*)

  (*) Value not defined in all devices.
- – LL_TIM_OCREF_CLR_INT_COMP2

Return values

- **None:**

Notes

- The OCxREF signal of a given channel can be cleared when a high level is applied on the OCREF_CLR_INPUT
- This function can only be used in Output compare and PWM modes.

Reference Manual to LL API cross reference:

- SMCR OCCS LL_TIM_SetOCRefClearInputSource
- AF2 OCRSEL LL_TIM_SetOCRefClearInputSource

LL_TIM_ClearFlag_UPDATE

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_UPDATE (TIM_TypeDef * TIMx)

Function description

Clear the update interrupt flag (UIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_ClearFlag_UPDATE

LL_TIM_IsActiveFlag_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicate whether update interrupt flag (UIF) is set (update interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR UIF LL_TIM_IsActiveFlag_UPDATE

LL_TIM_ClearFlag_CC1

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 interrupt flag (CC1F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_ClearFlag_CC1

LL_TIM_IsActiveFlag_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 interrupt flag (CC1F) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1IF LL_TIM_IsActiveFlag_CC1

LL_TIM_ClearFlag_CC2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 interrupt flag (CC2F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_ClearFlag_CC2

LL_TIM_IsActiveFlag_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 interrupt flag (CC2F) is set (Capture/Compare 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2IF LL_TIM_IsActiveFlag_CC2

LL_TIM_ClearFlag_CC3

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 interrupt flag (CC3F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_ClearFlag_CC3

LL_TIM_IsActiveFlag_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 interrupt flag (CC3F) is set (Capture/Compare 3 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3IF LL_TIM_IsActiveFlag_CC3

LL_TIM_ClearFlag_CC4

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 interrupt flag (CC4F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_ClearFlag_CC4

LL_TIM_IsActiveFlag_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 interrupt flag (CC4F) is set (Capture/Compare 4 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4IF LL_TIM_IsActiveFlag_CC4

LL_TIM_ClearFlag_CC5

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC5 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 5 interrupt flag (CC5F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_ClearFlag_CC5

LL_TIM_IsActiveFlag_CC5

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC5 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 5 interrupt flag (CC5F) is set (Capture/Compare 5 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC5IF LL_TIM_IsActiveFlag_CC5

LL_TIM_ClearFlag_CC6

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC6 (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 6 interrupt flag (CC6F).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_ClearFlag_CC6

LL_TIM_IsActiveFlag_CC6

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC6 (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 6 interrupt flag (CC6F) is set (Capture/Compare 6 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC6IF LL_TIM_IsActiveFlag_CC6

LL_TIM_ClearFlag_COM

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_COM (TIM_TypeDef * TIMx)

Function description

Clear the commutation interrupt flag (COMIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_ClearFlag_COM

LL_TIM_IsActiveFlag_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_COM (const TIM_TypeDef * TIMx)

Function description

Indicate whether commutation interrupt flag (COMIF) is set (commutation interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR COMIF LL_TIM_IsActiveFlag_COM

LL_TIM_ClearFlag_TRIG

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TRIG (TIM_TypeDef * TIMx)

Function description

Clear the trigger interrupt flag (TIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_ClearFlag_TRIG

LL_TIM_IsActiveFlag_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicate whether trigger interrupt flag (TIF) is set (trigger interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TIF LL_TIM_IsActiveFlag_TRIG

LL_TIM_ClearFlag_BRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK (TIM_TypeDef * TIMx)

Function description

Clear the break interrupt flag (BIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_ClearFlag_BRK

LL_TIM_IsActiveFlag_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether break interrupt flag (BIF) is set (break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BIF LL_TIM_IsActiveFlag_BRK

LL_TIM_ClearFlag_BRK2

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_BRK2 (TIM_TypeDef * TIMx)

Function description

Clear the break 2 interrupt flag (B2IF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_ClearFlag_BRK2

LL_TIM_IsActiveFlag_BRK2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_BRK2 (const TIM_TypeDef * TIMx)

Function description

Indicate whether break 2 interrupt flag (B2IF) is set (break 2 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR B2IF LL_TIM_IsActiveFlag_BRK2

LL_TIM_ClearFlag_CC1OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC1OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 1 over-capture interrupt flag (CC1OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_ClearFlag_CC1OVR

LL_TIM_IsActiveFlag_CC1OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC1OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 1 over-capture interrupt flag (CC1OF) is set (Capture/Compare 1 interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC1OF LL_TIM_IsActiveFlag_CC1OVR

LL_TIM_ClearFlag_CC2OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC2OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 2 over-capture interrupt flag (CC2OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_ClearFlag_CC2OVR

LL_TIM_IsActiveFlag_CC2OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC2OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 2 over-capture interrupt flag (CC2OF) is set (Capture/Compare 2 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC2OF LL_TIM_IsActiveFlag_CC2OVR

LL_TIM_ClearFlag_CC3OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC3OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 3 over-capture interrupt flag (CC3OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_ClearFlag_CC3OVR

LL_TIM_IsActiveFlag_CC3OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC3OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 3 over-capture interrupt flag (CC3OF) is set (Capture/Compare 3 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC3OF LL_TIM_IsActiveFlag_CC3OVR

LL_TIM_ClearFlag_CC4OVR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_CC4OVR (TIM_TypeDef * TIMx)

Function description

Clear the Capture/Compare 4 over-capture interrupt flag (CC4OF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_ClearFlag_CC4OVR

LL_TIM_IsActiveFlag_CC4OVR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_CC4OVR (const TIM_TypeDef * TIMx)

Function description

Indicate whether Capture/Compare 4 over-capture interrupt flag (CC4OF) is set (Capture/Compare 4 over-capture interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR CC4OF LL_TIM_IsActiveFlag_CC4OVR

LL_TIM_ClearFlag_SYSBRK

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_SYSBRK (TIM_TypeDef * TIMx)

Function description

Clear the system break interrupt flag (SBIF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_ClearFlag_SYSBRK

LL_TIM_IsActiveFlag_SYSBRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_SYSBRK (const TIM_TypeDef * TIMx)

Function description

Indicate whether system break interrupt flag (SBIF) is set (system break interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR SBIF LL_TIM_IsActiveFlag_SYSBRK

LL_TIM_ClearFlag_TERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_TERR (TIM_TypeDef * TIMx)

Function description

Clear the transition error interrupt flag (TERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_ClearFlag_TERR

LL_TIM_IsActiveFlag_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_TERR (const TIM_TypeDef * TIMx)

Function description

Indicate whether transition error interrupt flag (TERRF) is set (transition error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR TERRF LL_TIM_IsActiveFlag_TERR

LL_TIM_ClearFlag_IERR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IERR (TIM_TypeDef * TIMx)

Function description

Clear the index error interrupt flag (IERRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_ClearFlag_IERR

LL_TIM_IsActiveFlag_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IERR (const TIM_TypeDef * TIMx)

Function description

Indicate whether index error interrupt flag (IERRF) is set (index error interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- SR IERRF LL_TIM_IsActiveFlag_IERR

LL_TIM_ClearFlag_DIR

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_DIR (TIM_TypeDef * TIMx)

Function description

Clear the direction change interrupt flag (DIRF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_ClearFlag_DIR

LL_TIM_IsActiveFlag_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_DIR (const TIM_TypeDef * TIMx)

Function description

Indicate whether direction change interrupt flag (DIRF) is set (direction change interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR DIRF LL_TIM_IsActiveFlag_DIR

LL_TIM_ClearFlag_IDX

Function name

__STATIC_INLINE void LL_TIM_ClearFlag_IDX (TIM_TypeDef * TIMx)

Function description

Clear the index interrupt flag (IDXF).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_ClearFlag_IDX

LL_TIM_IsActiveFlag_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsActiveFlag_IDX (const TIM_TypeDef * TIMx)

Function description

Indicate whether index interrupt flag (IDXF) is set (index interrupt is pending).

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- SR IDXF LL_TIM_IsActiveFlag_IDX

LL_TIM_EnableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_EnableIT_UPDATE

LL_TIM_DisableIT_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableIT_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update interrupt (UIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_DisableIT_UPDATE

LL_TIM_IsEnabledIT_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update interrupt (UIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UIE LL_TIM_IsEnabledIT_UPDATE

LL_TIM_EnableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_EnableIT_CC1

LL_TIM_DisableIT_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 interrupt (CC1IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_DisableIT_CC1

LL_TIM_IsEnabledIT_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 interrupt (CC1IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1IE LL_TIM_IsEnabledIT_CC1

LL_TIM_EnableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_EnableIT_CC2

LL_TIM_DisableIT_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 interrupt (CC2IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_DisableIT_CC2

LL_TIM_IsEnabledIT_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 interrupt (CC2IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2IE LL_TIM_IsEnabledIT_CC2

LL_TIM_EnableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_EnableIT_CC3

LL_TIM_DisableIT_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 interrupt (CC3IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_DisableIT_CC3

LL_TIM_IsEnabledIT_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 interrupt (CC3IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3IE LL_TIM_IsEnabledIT_CC3

LL_TIM_EnableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_EnableIT_CC4

LL_TIM_DisableIT_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableIT_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 interrupt (CC4IE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_DisableIT_CC4

LL_TIM_IsEnabledIT_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 interrupt (CC4IE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4IE LL_TIM_IsEnabledIT_CC4

LL_TIM_EnableIT_COM

Function name

__STATIC_INLINE void LL_TIM_EnableIT_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_EnableIT_COM

LL_TIM_DisableIT_COM

Function name

__STATIC_INLINE void LL_TIM_DisableIT_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation interrupt (COMIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_DisableIT_COM

LL_TIM_IsEnabledIT_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation interrupt (COMIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMIE LL_TIM_IsEnabledIT_COM

LL_TIM_EnableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_EnableIT_TRIG

LL_TIM_DisableIT_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_DisableIT_TRIG

LL_TIM_IsEnabledIT_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TIE LL_TIM_IsEnabledIT_TRIG

LL_TIM_EnableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_EnableIT_BRK (TIM_TypeDef * TIMx)

Function description

Enable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_EnableIT_BRK

LL_TIM_DisableIT_BRK

Function name

__STATIC_INLINE void LL_TIM_DisableIT_BRK (TIM_TypeDef * TIMx)

Function description

Disable break interrupt (BIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_DisableIT_BRK

LL_TIM_IsEnabledIT_BRK

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_BRK (const TIM_TypeDef * TIMx)

Function description

Indicates whether the break interrupt (BIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER BIE LL_TIM_IsEnabledIT_BRK

LL_TIM_EnableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_TERR (TIM_TypeDef * TIMx)

Function description

Enable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_EnableIT_TERR

LL_TIM_DisableIT_TERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_TERR (TIM_TypeDef * TIMx)

Function description

Disable transition error interrupt (TERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_DisableIT_TERR

LL_TIM_IsEnabledIT_TERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_TERR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the transition error interrupt (TERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER TERRIE LL_TIM_IsEnabledIT_TERR

LL_TIM_EnableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IERR (TIM_TypeDef * TIMx)

Function description

Enable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_EnableIT_IERR

LL_TIM_DisableIT_IERR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IERR (TIM_TypeDef * TIMx)

Function description

Disable index error interrupt (IERRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_DisableIT_IERR

LL_TIM_IsEnabledIT_IERR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IERR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the index error interrupt (IERRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_ENCODER_ERROR_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder error management.

Reference Manual to LL API cross reference:

- DIER IERRIE LL_TIM_IsEnabledIT_IERR

LL_TIM_EnableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_EnableIT_DIR (TIM_TypeDef * TIMx)

Function description

Enable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_EnableIT_DIR

LL_TIM_DisableIT_DIR

Function name

__STATIC_INLINE void LL_TIM_DisableIT_DIR (TIM_TypeDef * TIMx)

Function description

Disable direction change interrupt (DIRIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_DisableIT_DIR

LL_TIM_IsEnabledIT_DIR

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_DIR (const TIM_TypeDef * TIMx)

Function description

Indicates whether the direction change interrupt (DIRIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER DIRIE LL_TIM_IsEnabledIT_DIR

LL_TIM_EnableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_EnableIT_IDX (TIM_TypeDef * TIMx)

Function description

Enable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_EnableIT_IDX

LL_TIM_DisableIT_IDX

Function name

__STATIC_INLINE void LL_TIM_DisableIT_IDX (TIM_TypeDef * TIMx)

Function description

Disable index interrupt (IDXIE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_DisableIT_IDX

LL_TIM_IsEnabledIT_IDX

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledIT_IDX (const TIM_TypeDef * TIMx)

Function description

Indicates whether the index interrupt (IDXIE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_TIM_FUNCTINONAL_ENCODER_INTERRUPT_INSTANCE(TIMx) can be used to check whether or not a timer instance provides encoder interrupt management.

Reference Manual to LL API cross reference:

- DIER IDXIE LL_TIM_IsEnabledIT_IDX

LL_TIM_EnableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Enable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_EnableDMAReq_UPDATE

LL_TIM_DisableDMAReq_UPDATE

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_UPDATE (TIM_TypeDef * TIMx)

Function description

Disable update DMA request (UDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_DisableDMAReq_UPDATE

LL_TIM_IsEnabledDMAReq_UPDATE

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_UPDATE (const TIM_TypeDef * TIMx)

Function description

Indicates whether the update DMA request (UDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER UDE LL_TIM_IsEnabledDMAReq_UPDATE

LL_TIM_EnableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_EnableDMAReq_CC1

LL_TIM_DisableDMAReq_CC1

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC1 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 1 DMA request (CC1DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_DisableDMAReq_CC1

LL_TIM_IsEnabledDMAReq_CC1

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC1 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 1 DMA request (CC1DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC1DE LL_TIM_IsEnabledDMAReq_CC1

LL_TIM_EnableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_EnableDMAReq_CC2

LL_TIM_DisableDMAReq_CC2

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC2 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 2 DMA request (CC2DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_DisableDMAReq_CC2

LL_TIM_IsEnabledDMAReq_CC2

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC2 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 2 DMA request (CC2DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC2DE LL_TIM_IsEnabledDMAReq_CC2

LL_TIM_EnableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_EnableDMAReq_CC3

LL_TIM_DisableDMAReq_CC3

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC3 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 3 DMA request (CC3DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_DisableDMAReq_CC3

LL_TIM_IsEnabledDMAReq_CC3

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC3 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 3 DMA request (CC3DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC3DE LL_TIM_IsEnabledDMAReq_CC3

LL_TIM_EnableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Enable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_EnableDMAReq_CC4

LL_TIM_DisableDMAReq_CC4

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_CC4 (TIM_TypeDef * TIMx)

Function description

Disable capture/compare 4 DMA request (CC4DE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_DisableDMAReq_CC4

LL_TIM_IsEnabledDMAReq_CC4

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_CC4 (const TIM_TypeDef * TIMx)

Function description

Indicates whether the capture/compare 4 DMA request (CC4DE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER CC4DE LL_TIM_IsEnabledDMAReq_CC4

LL_TIM_EnableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Enable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_EnableDMAReq_COM

LL_TIM_DisableDMAReq_COM

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_COM (TIM_TypeDef * TIMx)

Function description

Disable commutation DMA request (COMDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_DisableDMAReq_COM

LL_TIM_IsEnabledDMAReq_COM

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_COM (const TIM_TypeDef * TIMx)

Function description

Indicates whether the commutation DMA request (COMDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER COMDE LL_TIM_IsEnabledDMAReq_COM

LL_TIM_EnableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_EnableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Enable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_EnableDMAReq_TRIG

LL_TIM_DisableDMAReq_TRIG

Function name

__STATIC_INLINE void LL_TIM_DisableDMAReq_TRIG (TIM_TypeDef * TIMx)

Function description

Disable trigger interrupt (TDE).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_DisableDMAReq_TRIG

LL_TIM_IsEnabledDMAReq_TRIG

Function name

__STATIC_INLINE uint32_t LL_TIM_IsEnabledDMAReq_TRIG (const TIM_TypeDef * TIMx)

Function description

Indicates whether the trigger interrupt (TDE) is enabled.

Parameters

- **TIMx:** Timer instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- DIER TDE LL_TIM_IsEnabledDMAReq_TRIG

LL_TIM_GenerateEvent_UPDATE

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_UPDATE (TIM_TypeDef * TIMx)

Function description

Generate an update event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR UG LL_TIM_GenerateEvent_UPDATE

LL_TIM_GenerateEvent_CC1

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC1 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 1 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC1G LL_TIM_GenerateEvent_CC1

LL_TIM_GenerateEvent_CC2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC2 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC2G LL_TIM_GenerateEvent_CC2

LL_TIM_GenerateEvent_CC3

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC3 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 3 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC3G LL_TIM_GenerateEvent_CC3

LL_TIM_GenerateEvent_CC4

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_CC4 (TIM_TypeDef * TIMx)

Function description

Generate Capture/Compare 4 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR CC4G LL_TIM_GenerateEvent_CC4

LL_TIM_GenerateEvent_COM

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_COM (TIM_TypeDef * TIMx)

Function description

Generate commutation event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR COMG LL_TIM_GenerateEvent_COM

LL_TIM_GenerateEvent_TRIG

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_TRIG (TIM_TypeDef * TIMx)

Function description

Generate trigger event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR TG LL_TIM_GenerateEvent_TRIG

LL_TIM_GenerateEvent_BRK

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK (TIM_TypeDef * TIMx)

Function description

Generate break event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR BG LL_TIM_GenerateEvent_BRK

LL_TIM_GenerateEvent_BRK2

Function name

__STATIC_INLINE void LL_TIM_GenerateEvent_BRK2 (TIM_TypeDef * TIMx)

Function description

Generate break 2 event.

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- EGR B2G LL_TIM_GenerateEvent_BRK2

LL_TIM_DeInit

Function name

ErrorStatus LL_TIM_DeInit (const TIM_TypeDef * TIMx)

Function description

Set TIMx registers to their reset values.

Parameters

- **TIMx:** Timer instance

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx registers are de-initialized
  – ERROR: invalid TIMx instance

LL_TIM_StructInit

Function name

void LL_TIM_StructInit (LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Set the fields of the time base unit configuration data structure to their default values.

Parameters

- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (time base unit configuration data structure)

Return values

- **None:**

LL_TIM_Init

Function name

ErrorStatus LL_TIM_Init (TIM_TypeDef * TIMx, const LL_TIM_InitTypeDef * TIM_InitStruct)

Function description

Configure the TIMx time base unit.

Parameters

- **TIMx:** Timer Instance
- **TIM_InitStruct:** pointer to a LL_TIM_InitTypeDef structure (TIMx time base unit configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx registers are de-initialized
  – ERROR: not applicable

LL_TIM_OC_StructInit

Function name

void LL_TIM_OC_StructInit (LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Set the fields of the TIMx output channel configuration data structure to their default values.

Parameters

- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (the output channel configuration data structure)

Return values

- **None:**

LL_TIM_OC_Init

Function name

ErrorStatus LL_TIM_OC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_OC_InitTypeDef * TIM_OC_InitStruct)

Function description

Configure the TIMx output channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  - LL_TIM_CHANNEL_CH1
  - LL_TIM_CHANNEL_CH2
  - LL_TIM_CHANNEL_CH3
  - LL_TIM_CHANNEL_CH4
  - LL_TIM_CHANNEL_CH5
  - LL_TIM_CHANNEL_CH6
- **TIM_OC_InitStruct:** pointer to a LL_TIM_OC_InitTypeDef structure (TIMx output channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx output channel is initialized
  - ERROR: TIMx output channel is not initialized

LL_TIM_IC_StructInit

Function name

void LL_TIM_IC_StructInit (LL_TIM_IC_InitTypeDef * TIM_ICInitStruct)

Function description

Set the fields of the TIMx input channel configuration data structure to their default values.

Parameters

- **TIM_ICInitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (the input channel configuration data structure)

Return values

- **None:**

LL_TIM_IC_Init

Function name

ErrorStatus LL_TIM_IC_Init (TIM_TypeDef * TIMx, uint32_t Channel, const LL_TIM_IC_InitTypeDef * TIM_IC_InitStruct)

Function description

Configure the TIMx input channel.

Parameters

- **TIMx:** Timer Instance
- **Channel:** This parameter can be one of the following values:
  – LL_TIM_CHANNEL_CH1
  – LL_TIM_CHANNEL_CH2
  – LL_TIM_CHANNEL_CH3
  – LL_TIM_CHANNEL_CH4
- **TIM_IC_InitStruct:** pointer to a LL_TIM_IC_InitTypeDef structure (TIMx input channel configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: TIMx output channel is initialized
  – ERROR: TIMx output channel is not initialized

LL_TIM_ENCODER_StructInit

Function name

void LL_TIM_ENCODER_StructInit (LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Fills each TIM_EncoderInitStruct field with its default value.

Parameters

- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (encoder interface configuration data structure)

Return values

- **None:**

LL_TIM_ENCODER_Init

Function name

ErrorStatus LL_TIM_ENCODER_Init (TIM_TypeDef * TIMx, const LL_TIM_ENCODER_InitTypeDef * TIM_EncoderInitStruct)

Function description

Configure the encoder interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_EncoderInitStruct:** pointer to a LL_TIM_ENCODER_InitTypeDef structure (TIMx encoder interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

LL_TIM_HALLSENSOR_StructInit

Function name

void LL_TIM_HALLSENSOR_StructInit (LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Set the fields of the TIMx Hall sensor interface configuration data structure to their default values.

Parameters

- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (HALL sensor interface configuration data structure)

Return values

- **None:**

LL_TIM_HALLSENSOR_Init

Function name

ErrorStatus LL_TIM_HALLSENSOR_Init (TIM_TypeDef * TIMx, const LL_TIM_HALLSENSOR_InitTypeDef * TIM_HallSensorInitStruct)

Function description

Configure the Hall sensor interface of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_HallSensorInitStruct:** pointer to a LL_TIM_HALLSENSOR_InitTypeDef structure (TIMx HALL sensor interface configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: TIMx registers are de-initialized
  - ERROR: not applicable

Notes

- TIMx CH1, CH2 and CH3 inputs connected through a XOR to the TI1 input channel
- TIMx slave mode controller is configured in reset mode. Selected internal trigger is TI1F_ED.
- Channel 1 is configured as input, IC1 is mapped on TRC.
- Captured value stored in TIMx_CCR1 correspond to the time elapsed between 2 changes on the inputs. It gives information about motor speed.
- Channel 2 is configured in output PWM 2 mode.
- Compare value stored in TIMx_CCR2 corresponds to the commutation delay.
- OC2REF is selected as trigger output on TRGO.
- LL_TIM_IC_POLARITY_BOTHEDGE must not be used for TI1 when it is used when TIMx operates in Hall sensor interface mode.

LL_TIM_BDTR_StructInit

Function name

void LL_TIM_BDTR_StructInit (LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Set the fields of the Break and Dead Time configuration data structure to their default values.

Parameters

- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **None:**

LL_TIM_BDTR_Init

Function name

ErrorStatus LL_TIM_BDTR_Init (TIM_TypeDef * TIMx, const LL_TIM_BDTR_InitTypeDef * TIM_BDTRInitStruct)

Function description

Configure the Break and Dead Time feature of the timer instance.

Parameters

- **TIMx:** Timer Instance
- **TIM_BDTRInitStruct:** pointer to a LL_TIM_BDTR_InitTypeDef structure (Break and Dead Time configuration data structure)

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: Break and Dead Time is initialized
  - ERROR: not applicable

Notes

- As the bits BK2P, BK2E, BK2F[3:0], BKF[3:0], AOE, BKP, BKE, OSSI, OSSR and DTG[7:0] can be write-locked depending on the LOCK configuration, it can be necessary to configure all of them during the first write access to the TIMx_BDTR register.
- Macro IS_TIM_BREAK_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a break input.
- Macro IS_TIM_BKIN2_INSTANCE(TIMx) can be used to check whether or not a timer instance provides a second break input.

119.3 TIM Firmware driver defines

The following section lists the various define and macros of the module.

119.3.1 TIM

TIM

Active Input Selection

LL_TIM_ACTIVEINPUT_DIRECTTI

ICx is mapped on TIx

LL_TIM_ACTIVEINPUT_INDIRECTTI

ICx is mapped on TIy

LL_TIM_ACTIVEINPUT_TRC

ICx is mapped on TRC

Automatic output enable

LL_TIM_AUTOMATICOUTPUT_DISABLE

MOE can be set only by software

LL_TIM_AUTOMATICOUTPUT_ENABLE

MOE can be set by software or automatically at the next update event

BKIN POLARITY

LL_TIM_BKIN_POLARITY_LOW

BRK BKIN input is active low

LL_TIM_BKIN_POLARITY_HIGH

BRK BKIN input is active high

BKIN SOURCE

LL_TIM_BKIN_SOURCE_BKIN

BKIN input from AF controller

LL_TIM_BKIN_SOURCE_BKCOMP1

internal signal: COMP1 output

LL_TIM_BKIN_SOURCE_BKCOMP2

internal signal: COMP2 output

LL_TIM_BKIN_SOURCE_MDF1

internal signal: Digital filter break output

BREAK2 AF MODE

LL_TIM_BREAK2_AFMODE_INPUT

Break2 input BRK2 in input mode

LL_TIM_BREAK2_AFMODE_BIDIRECTIONAL

Break2 input BRK2 in bidirectional mode

Break2 Enable

LL_TIM_BREAK2_DISABLE

Break2 function disabled

LL_TIM_BREAK2_ENABLE

Break2 function enabled

BREAK2 FILTER

LL_TIM_BREAK2_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK2_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK2_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK2_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK2_FILTER_FDIV2_N6
fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK2_FILTER_FDIV2_N8
fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK2_FILTER_FDIV4_N6
fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK2_FILTER_FDIV4_N8
fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK2_FILTER_FDIV8_N6
fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK2_FILTER_FDIV8_N8
fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK2_FILTER_FDIV16_N5
fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK2_FILTER_FDIV16_N6
fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK2_FILTER_FDIV16_N8
fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK2_FILTER_FDIV32_N5
fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK2_FILTER_FDIV32_N6
fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK2_FILTER_FDIV32_N8
fSAMPLING=fDTS/32, N=8

BREAK2 POLARITY

LL_TIM_BREAK2_POLARITY_LOW
Break input BRK2 is active low

LL_TIM_BREAK2_POLARITY_HIGH
Break input BRK2 is active high

BREAK AF MODE

LL_TIM_BREAK_AFMODE_INPUT
Break input BRK in input mode

LL_TIM_BREAK_AFMODE_BIDIRECTIONAL
Break input BRK in bidirectional mode

Break Enable

LL_TIM_BREAK_DISABLE
Break function disabled

LL_TIM_BREAK_ENABLE

Break function enabled

break filter

LL_TIM_BREAK_FILTER_FDIV1

No filter, BRK acts asynchronously

LL_TIM_BREAK_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_BREAK_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_BREAK_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_BREAK_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_BREAK_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_BREAK_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_BREAK_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_BREAK_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_BREAK_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_BREAK_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_BREAK_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_BREAK_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_BREAK_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_BREAK_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_BREAK_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

BREAK INPUT

LL_TIM_BREAK_INPUT_BKIN

TIMx_BKIN input

LL_TIM_BREAK_INPUT_BKIN2

TIMx_BKIN2 input

break polarity

LL_TIM_BREAK_POLARITY_LOW

Break input BRK is active low

LL_TIM_BREAK_POLARITY_HIGH

Break input BRK is active high

Capture Compare DMA Request

LL_TIM_CCDMAREQUEST_CC

CCx DMA request sent when CCx event occurs

LL_TIM_CCDMAREQUEST_UPDATE

CCx DMA requests sent when update event occurs

Capture Compare Update Source

LL_TIM_CCUPDATESOURCE_COMG_ONLY

Capture/compare control bits are updated by setting the COMG bit only

LL_TIM_CCUPDATESOURCE_COMG_AND_TRGI

Capture/compare control bits are updated by setting the COMG bit or when a rising edge occurs on trigger input (TRGI)

Channel

LL_LPTIM_CHANNEL_CH1

LPTIM input/output channel 1

LL_LPTIM_CHANNEL_CH2

LPTIM input/output channel 2

LL_TIM_CHANNEL_CH1

Timer input/output channel 1

LL_TIM_CHANNEL_CH1N

Timer complementary output channel 1

LL_TIM_CHANNEL_CH2

Timer input/output channel 2

LL_TIM_CHANNEL_CH2N

Timer complementary output channel 2

LL_TIM_CHANNEL_CH3

Timer input/output channel 3

LL_TIM_CHANNEL_CH3N

Timer complementary output channel 3

LL_TIM_CHANNEL_CH4

Timer input/output channel 4

LL_TIM_CHANNEL_CH4N

Timer complementary output channel 4

LL_TIM_CHANNEL_CH5

Timer output channel 5

LL_TIM_CHANNEL_CH6

Timer output channel 6

Clock Division

LL_TIM_CLOCKDIVISION_DIV1

tDTS=tCK_INT

LL_TIM_CLOCKDIVISION_DIV2

tDTS=2*tCK_INT

LL_TIM_CLOCKDIVISION_DIV4

tDTS=4*tCK_INT

Clock Source

LL_TIM_CLOCKSOURCE_INTERNAL

The timer is clocked by the internal clock provided from the RCC

LL_TIM_CLOCKSOURCE_EXT_MODE1

Counter counts at each rising or falling edge on a selected input

LL_TIM_CLOCKSOURCE_EXT_MODE2

Counter counts at each rising or falling edge on the external trigger input ETR

Counter Direction

LL_TIM_COUNTERDIRECTION_UP

Timer counter counts up

LL_TIM_COUNTERDIRECTION_DOWN

Timer counter counts down

Counter Mode

LL_TIM_COUNTERMODE_UP

Counter used as upcounter

LL_TIM_COUNTERMODE_DOWN

Counter used as downcounter

LL_TIM_COUNTERMODE_CENTER_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting down.

LL_TIM_COUNTERMODE_CENTER_UP

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up

LL_TIM_COUNTERMODE_CENTER_UP_DOWN

The counter counts up and down alternatively. Output compare interrupt flags of output channels are set only when the counter is counting up or down.

DMA Burst Base Address

LL_TIM_DMABURST_BASEADDR_CR1

TIMx_CR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CR2

TIMx_CR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SMCR

TIMx_SMCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DIER

TIMx_DIER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_SR

TIMx_SR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_EGR

TIMx_EGR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR1

TIMx_CCMR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR2

TIMx_CCMR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCER

TIMx_CCER register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CNT

TIMx_CNT register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_PSC

TIMx_PSC register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ARR

TIMx_ARR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_RCR

TIMx_RCR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR1

TIMx_CCR1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR2

TIMx_CCR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR3

TIMx_CCR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR4

TIMx_CCR4 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_BDTR

TIMx_BDTR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR5

TIMx_CCR5 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCR6

TIMx_CCR6 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_CCMR3

TIMx_CCMR3 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_DTR2

TIMx_DTR2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_ECR

TIMx_ECR register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_TISEL

TIMx_TISEL register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF1

TIMx_AF1 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_AF2

TIMx_AF2 register is the DMA base address for DMA burst

LL_TIM_DMABURST_BASEADDR_OR1

TIMx_OR1 register is the DMA base address for DMA burst

DMA Burst Length

LL_TIM_DMABURST_LENGTH_1TRANSFER

Transfer is done to 1 register starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_2TRANSFERS

Transfer is done to 2 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_3TRANSFERS

Transfer is done to 3 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_4TRANSFERS

Transfer is done to 4 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_5TRANSFERS

Transfer is done to 5 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_6TRANSFERS

Transfer is done to 6 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_7TRANSFERS

Transfer is done to 7 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_8TRANSFERS

Transfer is done to 1 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_9TRANSFERS

Transfer is done to 9 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_10TRANSFERS

Transfer is done to 10 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_11TRANSFERS

Transfer is done to 11 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_12TRANSFERS

Transfer is done to 12 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_13TRANSFERS

Transfer is done to 13 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_14TRANSFERS

Transfer is done to 14 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_15TRANSFERS

Transfer is done to 15 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_16TRANSFERS

Transfer is done to 16 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_17TRANSFERS

Transfer is done to 17 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_18TRANSFERS

Transfer is done to 18 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_19TRANSFERS

Transfer is done to 19 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_20TRANSFERS

Transfer is done to 20 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_21TRANSFERS

Transfer is done to 21 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_22TRANSFERS

Transfer is done to 22 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_23TRANSFERS

Transfer is done to 23 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_24TRANSFERS

Transfer is done to 24 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_25TRANSFERS

Transfer is done to 25 registers starting from the DMA burst base address

LL_TIM_DMABURST_LENGTH_26TRANSFERS

Transfer is done to 26 registers starting from the DMA burst base address

DMA Burst Source

LL_TIM_DMA_UPDATE

Transfer source is update event

LL_TIM_DMA_CC1

Transfer source is CC1 event

LL_TIM_DMA_CC2

Transfer source is CC2 event

LL_TIM_DMA_CC3

Transfer source is CC3 event

LL_TIM_DMA_CC4

Transfer source is CC4 event

LL_TIM_DMA_COM

Transfer source is COM event

LL_TIM_DMA_TRIGGER

Transfer source is trigger event

Encoder Mode

LL_TIM_ENCODERMODE_X2_TI1

Quadrature encoder mode 1, x2 mode - Counter counts up/down on TI1FP1 edge depending on TI2FP2 level

LL_TIM_ENCODERMODE_X2_TI2

Quadrature encoder mode 2, x2 mode - Counter counts up/down on TI2FP2 edge depending on TI1FP1 level

LL_TIM_ENCODERMODE_X4_TI12

Quadrature encoder mode 3, x4 mode - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X2

Encoder mode: Clock plus direction - x2 mode

LL_TIM_ENCODERMODE_CLOCKPLUSDIRECTION_X1

Encoder mode: Clock plus direction, x1 mode, TI2FP2 edge sensitivity is set by CC2P

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X2

Encoder mode: Directional Clock, x2 mode

LL_TIM_ENCODERMODE_DIRECTIONALCLOCK_X1_TI12

Encoder mode: Directional Clock, x1 mode, TI1FP1 and TI2FP2 edge sensitivity is set by CC1P and CC2P

LL_TIM_ENCODERMODE_X1_TI1

Quadrature encoder mode: x1 mode, counting on TI1FP1 edges only, edge sensitivity is set by CC1P

LL_TIM_ENCODERMODE_X1_TI2

Quadrature encoder mode: x1 mode, counting on TI2FP2 edges only, edge sensitivity is set by CC1P

External Trigger Filter

LL_TIM_ETR_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_ETR_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_ETR_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_ETR_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_ETR_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_ETR_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_ETR_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_ETR_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_ETR_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_ETR_FILTER_FDIV8_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_ETR_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_ETR_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_ETR_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_ETR_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_ETR_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

External Trigger Polarity

LL_TIM_ETR_POLARITY_NONINVERTED

ETR is non-inverted, active at high level or rising edge

LL_TIM_ETR_POLARITY_INVERTED

ETR is inverted, active at low level or falling edge

External Trigger Prescaler

LL_TIM_ETR_PRESCALER_DIV1

ETR prescaler OFF

LL_TIM_ETR_PRESCALER_DIV2

ETR frequency is divided by 2

LL_TIM_ETR_PRESCALER_DIV4

ETR frequency is divided by 4

LL_TIM_ETR_PRESCALER_DIV8

ETR frequency is divided by 8

first index selection

LL_TIM_INDEX_ALL

Index is always active

LL_TIM_INDEX_FIRST_ONLY

The first Index only resets the counter

Get Flags Defines

LL_TIM_SR_UIF

Update interrupt flag

LL_TIM_SR_CC1IF

Capture/compare 1 interrupt flag

LL_TIM_SR_CC2IF

Capture/compare 2 interrupt flag

LL_TIM_SR_CC3IF

Capture/compare 3 interrupt flag

LL_TIM_SR_CC4IF

Capture/compare 4 interrupt flag

LL_TIM_SR_CC5IF

Capture/compare 5 interrupt flag

LL_TIM_SR_CC6IF

Capture/compare 6 interrupt flag

LL_TIM_SR_COMIF

COM interrupt flag

LL_TIM_SR_TIF

Trigger interrupt flag

LL_TIM_SR_BIF

Break interrupt flag

LL_TIM_SR_B2IF

Second break interrupt flag

LL_TIM_SR_CC1OF

Capture/Compare 1 overcapture flag

LL_TIM_SR_CC2OF

Capture/Compare 2 overcapture flag

LL_TIM_SR_CC3OF

Capture/Compare 3 overcapture flag

LL_TIM_SR_CC4OF

Capture/Compare 4 overcapture flag

LL_TIM_SR_SBIF

System Break interrupt flag

LL_TIM_SR_IDXF

Index interrupt flag

LL_TIM_SR_DIRF

Direction Change interrupt flag

LL_TIM_SR_IERRF

Index Error flag

LL_TIM_SR_TERRF

Transition Error flag

GROUPCH5

LL_TIM_GROUPCH5_NONE

No effect of OC5REF on OC1REFC, OC2REFC and OC3REFC

LL_TIM_GROUPCH5_OC1REFC

OC1REFC is the logical AND of OC1REFC and OC5REF

LL_TIM_GROUPCH5_OC2REFC

OC2REFC is the logical AND of OC2REFC and OC5REF

LL_TIM_GROUPCH5_OC3REFC

OC3REFC is the logical AND of OC3REFC and OC5REF

Input Configuration Prescaler

LL_TIM_ICPSC_DIV1

No prescaler, capture is done each time an edge is detected on the capture input

LL_TIM_ICPSC_DIV2

Capture is done once every 2 events

LL_TIM_ICPSC_DIV4

Capture is done once every 4 events

LL_TIM_ICPSC_DIV8

Capture is done once every 8 events

Input Configuration Filter

LL_TIM_IC_FILTER_FDIV1

No filter, sampling is done at fDTS

LL_TIM_IC_FILTER_FDIV1_N2

fSAMPLING=fCK_INT, N=2

LL_TIM_IC_FILTER_FDIV1_N4

fSAMPLING=fCK_INT, N=4

LL_TIM_IC_FILTER_FDIV1_N8

fSAMPLING=fCK_INT, N=8

LL_TIM_IC_FILTER_FDIV2_N6

fSAMPLING=fDTS/2, N=6

LL_TIM_IC_FILTER_FDIV2_N8

fSAMPLING=fDTS/2, N=8

LL_TIM_IC_FILTER_FDIV4_N6

fSAMPLING=fDTS/4, N=6

LL_TIM_IC_FILTER_FDIV4_N8

fSAMPLING=fDTS/4, N=8

LL_TIM_IC_FILTER_FDIV8_N6

fSAMPLING=fDTS/8, N=6

LL_TIM_IC_FILTER_FDIV8_N8

fSAMPLING=fDTS/8, N=8

LL_TIM_IC_FILTER_FDIV16_N5

fSAMPLING=fDTS/16, N=5

LL_TIM_IC_FILTER_FDIV16_N6

fSAMPLING=fDTS/16, N=6

LL_TIM_IC_FILTER_FDIV16_N8

fSAMPLING=fDTS/16, N=8

LL_TIM_IC_FILTER_FDIV32_N5

fSAMPLING=fDTS/32, N=5

LL_TIM_IC_FILTER_FDIV32_N6

fSAMPLING=fDTS/32, N=6

LL_TIM_IC_FILTER_FDIV32_N8

fSAMPLING=fDTS/32, N=8

Input Configuration Polarity

LL_TIM_IC_POLARITY_RISING

The circuit is sensitive to TIxFP1 rising edge, TIxFP1 is not inverted

LL_TIM_IC_POLARITY_FALLING

The circuit is sensitive to TIxFP1 falling edge, TIxFP1 is inverted

LL_TIM_IC_POLARITY_BOTHEDGE

The circuit is sensitive to both TIxFP1 rising and falling edges, TIxFP1 is not inverted

index blanking selection

LL_TIM_INDEX_BLANK_ALWAYS

Index always active

LL_TIM_INDEX_BLANK_TI3

Index disabled when TI3 input is active, as per CC3P bitfield

LL_TIM_INDEX_BLANK_TI4

Index disabled when TI4 input is active, as per CC4P bitfield

index direction selection

LL_TIM_INDEX_UP_DOWN

Index resets the counter whatever the direction

LL_TIM_INDEX_UP

Index resets the counter when up-counting only

LL_TIM_INDEX_DOWN

Index resets the counter when down-counting only

index positioning selection

LL_TIM_INDEX_POSITION_DOWN_DOWN

Index resets the counter when AB = 00

LL_TIM_INDEX_POSITION_DOWN_UP

Index resets the counter when AB = 01

LL_TIM_INDEX_POSITION_UP_DOWN

Index resets the counter when AB = 10

LL_TIM_INDEX_POSITION_UP_UP

Index resets the counter when AB = 11

LL_TIM_INDEX_POSITION_DOWN

Index resets the counter when clock is 0

LL_TIM_INDEX_POSITION_UP

Index resets the counter when clock is 1

IT Defines

LL_TIM_DIER_UIE

Update interrupt enable

LL_TIM_DIER_CC1IE

Capture/compare 1 interrupt enable

LL_TIM_DIER_CC2IE

Capture/compare 2 interrupt enable

LL_TIM_DIER_CC3IE

Capture/compare 3 interrupt enable

LL_TIM_DIER_CC4IE

Capture/compare 4 interrupt enable

LL_TIM_DIER_COMIE

COM interrupt enable

LL_TIM_DIER_TIE

Trigger interrupt enable

LL_TIM_DIER_BIE

Break interrupt enable

LL_TIM_DIER_IDXIE

Index interrupt enable

LL_TIM_DIER_DIRIE

Direction Change interrupt enable

LL_TIM_DIER_IERRIE

Index Error interrupt enable

LL_TIM_DIER_TERRIE

Transition Error interrupt enable

Lock Level

LL_TIM_LOCKLEVEL_OFF

LOCK OFF - No bit is write protected

LL_TIM_LOCKLEVEL_1

LOCK Level 1

LL_TIM_LOCKLEVEL_2

LOCK Level 2

LL_TIM_LOCKLEVEL_3

LOCK Level 3

Output Configuration Idle State

LL_TIM_OCIDLESTATE_LOW

OCx=0 (after a dead-time if OC is implemented) when MOE=0

LL_TIM_OCIDLESTATE_HIGH

OCx=1 (after a dead-time if OC is implemented) when MOE=0

Output Configuration Mode

LL_TIM_OCMODE_FROZEN

The comparison between the output compare register TIMx_CCRy and the counter TIMx_CNT has no effect on the output channel level

LL_TIM_OCMODE_ACTIVE

OCyREF is forced high on compare match

LL_TIM_OCMODE_INACTIVE

OCyREF is forced low on compare match

LL_TIM_OCMODE_TOGGLE

OCyREF toggles on compare match

LL_TIM_OCMODE_FORCED_INACTIVE

OCyREF is forced low

LL_TIM_OCMODE_FORCED_ACTIVE

OCyREF is forced high

LL_TIM_OCMODE_PWM1

In upcounting, channel y is active as long as TIMx_CNT<TIMx_CCRy else inactive. In downcounting, channel y is inactive as long as TIMx_CNT>TIMx_CCRy else active.

LL_TIM_OCMODE_PWM2

In upcounting, channel y is inactive as long as TIMx_CNT<TIMx_CCRy else active. In downcounting, channel y is active as long as TIMx_CNT>TIMx_CCRy else inactive

LL_TIM_OCMODE_RETRIG_OPM1

Retrigerrable OPM mode 1

LL_TIM_OCMODE_RETRIG_OPM2

Retrigerrable OPM mode 2

LL_TIM_OCMODE_COMBINED_PWM1

Combined PWM mode 1

LL_TIM_OCMODE_COMBINED_PWM2

Combined PWM mode 2

LL_TIM_OCMODE_ASYMMETRIC_PWM1

Asymmetric PWM mode 1

LL_TIM_OCMODE_ASYMMETRIC_PWM2

Asymmetric PWM mode 2

LL_TIM_OCMODE_PULSE_ON_COMPARE

Pulse on Compare mode

LL_TIM_OCMODE_DIRECTION_OUTPUT

Direction output mode

Output Configuration Polarity

LL_TIM_OCPOLARITY_HIGH

OCxactive high

LL_TIM_OCPOLARITY_LOW

OCxactive low

OCREF clear input selection

LL_TIM_OCREF_CLR_INT_ETR

OCREF_CLR_INT is connected to ETRF

LL_TIM_OCREF_CLR_INT_COMP1

OCREF clear input is connected to COMP1_OUT

LL_TIM_OCREF_CLR_INT_COMP2

OCREF clear input is connected to COMP2_OUT

Output Configuration State

LL_TIM_OCSTATE_DISABLE

OCx is not active

LL_TIM_OCSTATE_ENABLE

OCx signal is output on the corresponding output pin

One Pulse Mode

LL_TIM_ONEPULSEMODE_SINGLE

Counter stops counting at the next update event

LL_TIM_ONEPULSEMODE_REPETITIVE

Counter is not stopped at update event

OSSI

LL_TIM_OSSI_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSI_ENABLE

When inactive, OxC/OCxN outputs are first forced with their inactive level then forced to their idle level after the deadtime

OSSR

LL_TIM_OSSR_DISABLE

When inactive, OCx/OCxN outputs are disabled

LL_TIM_OSSR_ENABLE

When inactive, OC/OCN outputs are enabled with their inactive level as soon as CCxE=1 or CCxNE=1

Pulse on compare pulse width prescaler

LL_TIM_PWPRSC_X1

Pulse on compare pulse width prescaler 1

LL_TIM_PWPRSC_X2

Pulse on compare pulse width prescaler 2

LL_TIM_PWPRSC_X4

Pulse on compare pulse width prescaler 4

LL_TIM_PWPRSC_X8

Pulse on compare pulse width prescaler 8

LL_TIM_PWPRSC_X16

Pulse on compare pulse width prescaler 16

LL_TIM_PWPRSC_X32

Pulse on compare pulse width prescaler 32

LL_TIM_PWPRSC_X64

Pulse on compare pulse width prescaler 64

LL_TIM_PWPRSC_X128

Pulse on compare pulse width prescaler 128

Slave Mode

LL_TIM_SLAVEMODE_DISABLED

Slave mode disabled

LL_TIM_SLAVEMODE_RESET

Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter

LL_TIM_SLAVEMODE_GATED

Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high

LL_TIM_SLAVEMODE_TRIGGER

Trigger Mode - The counter starts at a rising edge of the trigger TRGI

LL_TIM_SLAVEMODE_COMBINED_RESETTRIGGER

Combined reset + trigger mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter, generates an update of the registers and starts the counter

LL_TIM_SLAVEMODE_COMBINED_GATEDRESET

Combined gated + reset mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops and is reset) as soon as the trigger becomes low.Both startand stop of the counter are controlled.

SMS Preload Source

LL_TIM_SMSPS_TIMUPDATE

The SMS preload transfer is triggered by the Timer's Update event

LL_TIM_SMSPS_INDEX

The SMS preload transfer is triggered by the Index event

TIM15 External Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO

TIM15 TI1 is connected to GPIO

LL_TIM_TIM15_TI1_RMP_LSE

TIM15 TI1 is connected to LSE

LL_TIM_TIM15_TI1_RMP_COMP1

TIM15 TI1 is connected to COMP1 output

LL_TIM_TIM15_TI1_RMP_COMP2

TIM15 TI1 is connected to COMP2 output

TIM15 External Input Ch2 Remap

LL_TIM_TIM15_TI2_RMP_GPIO

TIM15 TI2 is connected to GPIO

LL_TIM_TIM15_TI2_RMP_COMP2

TIM15 TI2 is connected to COMP2 output

TIM16 External Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO

TIM16 TI1 is connected to GPIO

LL_TIM_TIM16_TI1_RMP_MCO

TIM16 TI1 is connected to MCO

LL_TIM_TIM16_TI1_RMP_HSE_DIV32

TIM16 TI1 is connected to HSE DIV32

LL_TIM_TIM16_TI1_RMP_RTC

TIM16 TI1 is connected to RTC

LL_TIM_TIM16_TI1_RMP_LSE

TIM16 TI1 is connected to LSE

LL_TIM_TIM16_TI1_RMP_LSI

TIM16 TI1 is connected to LSI

LL_TIM_TIM16_TI1_RMP_MSIS_1024

TIM16 TI1 is connected to MSIS/1024

LL_TIM_TIM16_TI1_RMP_MSIS_4

TIM16 TI1 is connected to MSIS/4

LL_TIM_TIM16_TI1_RMP_HSI_256

TIM16 TI1 is connected to HSI/256

TIM17 External Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17 TI1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_MCO

TIM17 TI1 is connected to MCO

LL_TIM_TIM17_TI1_RMP_HSE_DIV32

TIM17 TI1 is connected to HSE DIV32

LL_TIM_TIM17_TI1_RMP_RTC

TIM17 TI1 is connected to RTC

LL_TIM_TIM17_TI1_RMP_LSE

TIM17 TI1 is connected to LSE

LL_TIM_TIM17_TI1_RMP_LSI

TIM17 TI1 is connected to LSI

LL_TIM_TIM17_TI1_RMP_MSIS_1024

TIM17 TI1 is connected to MSIS/1024

LL_TIM_TIM17_TI1_RMP_MSIS_4

TIM17 TI1 is connected to MSIS/4

LL_TIM_TIM17_TI1_RMP_HSI_256

TIM17 TI1 is connected to HSI/256

External Trigger Source TIM1

LL_TIM_TIM1_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM1_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM1_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM1_ETRSOURCE_MSIK

ETR input is connected to MSIK

LL_TIM_TIM1_ETRSOURCE_HSI

ETR input is connected to HSI

LL_TIM_TIM1_ETRSOURCE_MSIS

ETR input is connected to MSIS

LL_TIM_TIM1_ETRSOURCE_ADC2_AWD2

ADC2 analog watchdog 2

LL_TIM_TIM1_ETRSOURCE_ADC2_AWD3

ADC2 analog watchdog 3

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD1

ADC1 analog watchdog 1

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD2

ADC1 analog watchdog 2

LL_TIM_TIM1_ETRSOURCE_ADC1_AWD3

ADC1 analog watchdog 3

LL_TIM_TIM1_ETRSOURCE_ADC4_AWD1

ADC4 analog watchdog 1

LL_TIM_TIM1_ETRSOURCE_ADC4_AWD2

ADC4 analog watchdog 2

LL_TIM_TIM1_ETRSOURCE_ADC4_AWD3

ADC4 analog watchdog 3

LL_TIM_TIM1_ETRSOURCE_ADC2_AWD1

ADC2 analog watchdog 1

TIM1 External Input Ch1 Remap

LL_TIM_TIM1_TI1_RMP_GPIO

TIM1 TI1 is connected to GPIO

LL_TIM_TIM1_TI1_RMP_COMP1

TIM1 TI1 is connected to COMP1 output

LL_TIM_TIM1_TI1_RMP_COMP2

TIM1 TI1 is connected to COMP2 output

External Trigger Source TIM2

LL_TIM_TIM2_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM2_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM2_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM2_ETRSOURCE_MSIK

ETR input is connected to MSIK

LL_TIM_TIM2_ETRSOURCE_HSI

ETR input is connected to HSI

LL_TIM_TIM2_ETRSOURCE_MSIS

ETR input is connected to MSIS

LL_TIM_TIM2_ETRSOURCE_DCMI_VSYNC

ETR input is connected to DCMI VSYNC

LL_TIM_TIM2_ETRSOURCE_LTDC_VSYNC
ETR input is connected to LTDC_VSYNC

LL_TIM_TIM2_ETRSOURCE_TIM3_ETR
ETR input is connected to TIM3 ETR

LL_TIM_TIM2_ETRSOURCE_TIM4_ETR
ETR input is connected to TIM4 ETR

LL_TIM_TIM2_ETRSOURCE_TIM5_ETR
ETR input is connected to TIM5 ETR

LL_TIM_TIM2_ETRSOURCE_LSE
ETR input is connected to LSE

LL_TIM_TIM2_ETRSOURCE_DSI_TE
ETR input is connected to DSI_TE

LL_TIM_TIM2_ETRSOURCE_DCMI_HSYNC
ETR input is connected to DCMI HSYNC

LL_TIM_TIM2_ETRSOURCE_LTDC_HSYNC
ETR input is connected to LTDC HSYNC

TIM2 External Input Ch1 Remap

LL_TIM_TIM2_TI1_RMP_GPIO
TIM2 TI1 is connected to GPIO

LL_TIM_TIM2_TI1_RMP_COMP1
TIM2 TI1 is connected to COMP1 output

LL_TIM_TIM2_TI1_RMP_COMP2
TIM2 TI1 is connected to COMP2 output

TIM16 External Input Ch2 Remap

LL_TIM_TIM2_TI2_RMP_GPIO
TIM2 TI2 is connected to GPIO

LL_TIM_TIM2_TI2_RMP_COMP1
TIM2 TI2 is connected to COMP1 output

LL_TIM_TIM2_TI2_RMP_COMP2
TIM2 TI2 is connected to COMP2 output

TIM2 External Input Ch4 Remap

LL_TIM_TIM2_TI4_RMP_GPIO
TIM2 TI4 is connected to COMP1 output

LL_TIM_TIM2_TI4_RMP_COMP1
TIM2 TI4 is connected to COMP1 output

LL_TIM_TIM2_TI4_RMP_COMP2
TIM2 TI4 is connected to COMP2 output

External Trigger Source TIM3

LL_TIM_TIM3_ETRSOURCE_GPIO
ETR input is connected to GPIO

LL_TIM_TIM3_ETRSOURCE_COMP1
ETR input is connected to COMP1_OUT

LL_TIM_TIM3_ETRSOURCE_COMP2
ETR input is connected to COMP2_OUT

LL_TIM_TIM3_ETRSOURCE_MSIK
ETR input is connected to MSIK

LL_TIM_TIM3_ETRSOURCE_HSI
ETR input is connected to HSI

LL_TIM_TIM3_ETRSOURCE_MSIS
ETR input is connected to MSIS

LL_TIM_TIM3_ETRSOURCE_DCMI_VSYNC
ETR input is connected to DCMI VSYNC

LL_TIM_TIM3_ETRSOURCE_LTDC_VSYNC
ETR input is connected to LTDC_VSYNC

LL_TIM_TIM3_ETRSOURCE_TIM2_ETR
ETR input is connected to TIM2 ETR

LL_TIM_TIM3_ETRSOURCE_TIM4_ETR
ETR input is connected to TIM4 ETR

LL_TIM_TIM3_ETRSOURCE_DSI_TE
ETR input is connected to DSI_TE

LL_TIM_TIM3_ETRSOURCE_ADC1_AWD1
ADC1 analog watchdog 1

LL_TIM_TIM3_ETRSOURCE_ADC1_AWD2
ADC1 analog watchdog 2

LL_TIM_TIM3_ETRSOURCE_ADC1_AWD3
ADC1 analog watchdog 3

LL_TIM_TIM3_ETRSOURCE_DCMI_HSYNC
ETR input is connected to DCMI HSYNC

LL_TIM_TIM3_ETRSOURCE_LTDC_HSYNC
ETR input is connected to LTDC HSYNC

TIM3 External Input Ch1 Remap

LL_TIM_TIM3_TI1_RMP_GPIO
TIM3 TI1 is connected to GPIO

LL_TIM_TIM3_TI1_RMP_COMP1
TIM3 TI1 is connected to COMP1 output

LL_TIM_TIM3_TI1_RMP_COMP2
TIM3 TI1 is connected to COMP2 output

TIM3 External Input Ch2 Remap

LL_TIM_TIM3_TI2_RMP_GPIO

TIM3 TI2 is connected to GPIO

LL_TIM_TIM3_TI2_RMP_COMP1

TIM3 TI2 is connected to COMP1 output

LL_TIM_TIM3_TI2_RMP_COMP2

TIM3 TI2 is connected to COMP2 output

External Trigger Source TIM4

LL_TIM_TIM4_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM4_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM4_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM4_ETRSOURCE_MSIK

ETR input is connected to MSIK

LL_TIM_TIM4_ETRSOURCE_HSI

ETR input is connected to HSI

LL_TIM_TIM4_ETRSOURCE_MSIS

ETR input is connected to MSIS

LL_TIM_TIM4_ETRSOURCE_DCMI_VSYNC

ETR input is connected to DCMI VSYNC

LL_TIM_TIM4_ETRSOURCE_LTDC_VSYNC

ETR input is connected to LTDC_VSYNC

LL_TIM_TIM4_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM4_ETRSOURCE_TIM5_ETR

ETR input is connected to TIM5 ETR

LL_TIM_TIM4_ETRSOURCE_DSI_TE

TIM2_ETR is connected to DSI_TE

LL_TIM_TIM4_ETRSOURCE_ADC2_AWD1

ETR input is connected to ADC2 AWD1

LL_TIM_TIM4_ETRSOURCE_ADC2_AWD2

ETR input is connected to ADC2 AWD2

LL_TIM_TIM4_ETRSOURCE_ADC2_AWD3

ETR input is connected to ADC2 AWD3

LL_TIM_TIM4_ETRSOURCE_DCMI_HSYNC

ETR input is connected to DCMI HSYNC

LL_TIM_TIM4_ETRSOURCE_LTDC_HSYNC

ETR input is connected to LTDC HSYNC

TIM4 External Input Ch1 Remap

LL_TIM_TIM4_TI1_RMP_GPIO

TIM4 TI1 is connected to GPIO

LL_TIM_TIM4_TI1_RMP_COMP1

TIM4 TI1 is connected to COMP1 output

LL_TIM_TIM4_TI1_RMP_COMP2

TIM4 TI1 is connected to COMP2 output

TIM4 External Input Ch2 Remap

LL_TIM_TIM4_TI2_RMP_GPIO

TIM4 TI2 is connected to GPIO

LL_TIM_TIM4_TI2_RMP_COMP1

TIM4 TI2 is connected to COMP1 output

LL_TIM_TIM4_TI2_RMP_COMP2

TIM4 TI2 is connected to COMP2 output

External Trigger Source TIM5

LL_TIM_TIM5_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM5_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM5_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM5_ETRSOURCE_MSIK

ETR input is connected to MSIK

LL_TIM_TIM5_ETRSOURCE_HSI

ETR input is connected to HSI

LL_TIM_TIM5_ETRSOURCE_MSIS

ETR input is connected to MSIS

LL_TIM_TIM5_ETRSOURCE_DCMI_VSYNC

ETR input is connected to DCMI VSYNC

LL_TIM_TIM5_ETRSOURCE_LTDC_VSYNC

ETR input is connected to LTDC_VSYNC

LL_TIM_TIM5_ETRSOURCE_TIM2_ETR

ETR input is connected to TIM2 ETR

LL_TIM_TIM5_ETRSOURCE_TIM3_ETR

ETR input is connected to TIM3 ETR

LL_TIM_TIM5_ETRSOURCE_DSI_TE

ETR input is connected to DSI_TE

LL_TIM_TIM5_ETRSOURCE_DCMI_HSYNC

ETR input is connected to DCMI HSYNC

LL_TIM_TIM5_ETRSOURCE_LTDC_HSYNC

ETR input is connected to LTDC HSYNC

TIM5 External Input Ch1 Remap

LL_TIM_TIM5_TI1_RMP_GPIO

TIM5 TI1 is connected to GPIO

LL_TIM_TIM5_TI1_RMP_LSI

TIM5 TI1 is connected to LSI

LL_TIM_TIM5_TI1_RMP_LSE

TIM5 TI1 is connected to LSE

LL_TIM_TIM5_TI1_RMP_RTC

TIM5 TI1 is connected to RTC Wakeup

LL_TIM_TIM5_TI1_RMP_COMP1

TIM5 TI1 is connected to COMP1 output

LL_TIM_TIM5_TI1_RMP_COMP2

TIM5 TI1 is connected to COMP2 output

TIM5 External Input Ch2 Remap

LL_TIM_TIM5_TI2_RMP_GPIO

TIM5 TI2 is connected to GPIO

LL_TIM_TIM5_TI2_RMP_COMP1

TIM5 TI2 is connected to COMP1 output

LL_TIM_TIM5_TI2_RMP_COMP2

TIM5 TI2 is connected to COMP2 output

External Trigger Source TIM8

LL_TIM_TIM8_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_TIM8_ETRSOURCE_COMP1

ETR input is connected to COMP1_OUT

LL_TIM_TIM8_ETRSOURCE_COMP2

ETR input is connected to COMP2_OUT

LL_TIM_TIM8_ETRSOURCE_MSIK

ETR input is connected to MSIK

LL_TIM_TIM8_ETRSOURCE_HSI

ETR input is connected to HSI

LL_TIM_TIM8_ETRSOURCE_MSIS
ETR input is connected to MSIS

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD2
ETR input is connected to ADC2 AWD2

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD3
ETR input is connected to ADC2 AWD3

LL_TIM_TIM8_ETRSOURCE_ADC1_AWD1
ADC1 analog watchdog 1

LL_TIM_TIM8_ETRSOURCE_ADC1_AWD2
ADC1 analog watchdog 2

LL_TIM_TIM8_ETRSOURCE_ADC1_AWD3
ADC1 analog watchdog 3

LL_TIM_TIM8_ETRSOURCE_ADC4_AWD1
ADC4 analog watchdog 1

LL_TIM_TIM8_ETRSOURCE_ADC4_AWD2
ADC4 analog watchdog 2

LL_TIM_TIM8_ETRSOURCE_ADC4_AWD3
ADC4 analog watchdog 3

LL_TIM_TIM8_ETRSOURCE_ADC2_AWD1
ETR input is connected to ADC2 AWD1

TIM8 External Input Ch1 Remap

LL_TIM_TIM8_TI1_RMP_GPIO
TIM8 TI1 is connected to GPIO

LL_TIM_TIM8_TI1_RMP_COMP1
TIM8 TI1 is connected to COMP1 output

LL_TIM_TIM8_TI1_RMP_COMP2
TIM8 TI1 is connected to COMP2 output

Trigger Output

LL_TIM_TRGO_RESET
UG bit from the TIMx_EGR register is used as trigger output

LL_TIM_TRGO_ENABLE
Counter Enable signal (CNT_EN) is used as trigger output

LL_TIM_TRGO_UPDATE
Update event is used as trigger output

LL_TIM_TRGO_CC1IF
CC1 capture or a compare match is used as trigger output

LL_TIM_TRGO_OC1REF
OC1REF signal is used as trigger output

LL_TIM_TRGO_OC2REF

OC2REF signal is used as trigger output

LL_TIM_TRGO_OC3REF

OC3REF signal is used as trigger output

LL_TIM_TRGO_OC4REF

OC4REF signal is used as trigger output

LL_TIM_TRGO_ENCODERCLK

Encoder clock signal is used as trigger output

Trigger Output 2

LL_TIM_TRGO2_RESET

UG bit from the TIMx_EGR register is used as trigger output 2

LL_TIM_TRGO2_ENABLE

Counter Enable signal (CNT_EN) is used as trigger output 2

LL_TIM_TRGO2_UPDATE

Update event is used as trigger output 2

LL_TIM_TRGO2_CC1F

CC1 capture or a compare match is used as trigger output 2

LL_TIM_TRGO2_OC1

OC1REF signal is used as trigger output 2

LL_TIM_TRGO2_OC2

OC2REF signal is used as trigger output 2

LL_TIM_TRGO2_OC3

OC3REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4

OC4REF signal is used as trigger output 2

LL_TIM_TRGO2_OC5

OC5REF signal is used as trigger output 2

LL_TIM_TRGO2_OC6

OC6REF signal is used as trigger output 2

LL_TIM_TRGO2_OC4_RISINGFALLING

OC4REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC6_RISINGFALLING

OC6REF rising or falling edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_RISING

OC4REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC4_RISING_OC6_FALLING

OC4REF rising or OC6REF falling edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_RISING

OC5REF or OC6REF rising edges are used as trigger output 2

LL_TIM_TRGO2_OC5_RISING_OC6_FALLING

OC5REF rising or OC6REF falling edges are used as trigger output 2

Trigger Selection

LL_TIM_TS_ITR0

Internal Trigger 0 (ITR0) is used as trigger input

LL_TIM_TS_ITR1

Internal Trigger 1 (ITR1) is used as trigger input

LL_TIM_TS_ITR2

Internal Trigger 2 (ITR2) is used as trigger input

LL_TIM_TS_ITR3

Internal Trigger 3 (ITR3) is used as trigger input

LL_TIM_TS_ITR4

Internal Trigger 4 (ITR4) is used as trigger input

LL_TIM_TS_ITR5

Internal Trigger 5 (ITR5) is used as trigger input

LL_TIM_TS_ITR6

Internal Trigger 6 (ITR6) is used as trigger input

LL_TIM_TS_ITR7

Internal Trigger 7 (ITR7) is used as trigger input

LL_TIM_TS_ITR8

Internal Trigger 8 (ITR8) is used as trigger input

LL_TIM_TS_ITR11

Internal Trigger 11 (ITR11) is used as trigger input

LL_TIM_TS_TI1F_ED

TI1 Edge Detector (TI1F_ED) is used as trigger input

LL_TIM_TS_TI1FP1

Filtered Timer Input 1 (TI1FP1) is used as trigger input

LL_TIM_TS_TI2FP2

Filtered Timer Input 2 (TI12P2) is used as trigger input

LL_TIM_TS_ETRF

Filtered external Trigger (ETRF) is used as trigger input

Update Source

LL_TIM_UPDATESOURCE_REGULAR

Counter overflow/underflow, Setting the UG bit or Update generation through the slave mode controller generates an update request

LL_TIM_UPDATESOURCE_COUNTER

Only counter overflow/underflow generates an update request

Common Write and read registers Macros

LL_TIM_WriteReg

Description:
- Write a value in TIM register.

Parameters:
- __INSTANCE__: TIM Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_TIM_ReadReg

Description:
- Read a value in TIM register.

Parameters:
- __INSTANCE__: TIM Instance
- __REG__: Register to be read

Return value:
- Register: value

TIM Exported Macros

__LL_TIM_GETFLAG_UIFCPY

Description:
- HELPER macro retrieving the UIFCPY flag from the counter value.

Parameters:
- __CNT__: Counter value

Return value:
- UIF: status bit

Notes:
- ex: __LL_TIM_GETFLAG_UIFCPY (LL_TIM_GetCounter ()); Relevant only if UIF flag remapping has been enabled (UIF status bit is copied to TIMx_CNT register bit 31)

__LL_TIM_CALC_DEADTIME

Description:
- HELPER macro calculating DTG[0:7] in the TIMx_BDTR register to achieve the requested dead time duration.

Parameters:
- __TIMCLK__: timer input clock frequency (in Hz)
- __CKD__: This parameter can be one of the following values:
  - LL_TIM_CLOCKDIVISION_DIV1
  - LL_TIM_CLOCKDIVISION_DIV2
  - LL_TIM_CLOCKDIVISION_DIV4
- __DT__: deadtime duration (in ns)

Return value:
- DTG[0:7]

Notes:
- ex: __LL_TIM_CALC_DEADTIME (80000000, LL_TIM_GetClockDivision (), 120);

__LL_TIM_CALC_PSC

Description:

- HELPER macro calculating the prescaler value to achieve the required counter clock frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __CNTCLK__: counter clock frequency (in Hz)

Return value:

- Prescaler: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PSC (80000000, 1000000);

__LL_TIM_CALC_ARR

Description:

- HELPER macro calculating the auto-reload value to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_ARR_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required output signal frequency.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __FREQ__: output signal frequency (in Hz)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_ARR_DITHER (1000000, LL_TIM_GetPrescaler (), 10000);

__LL_TIM_CALC_DELAY

Description:

- HELPER macro calculating the compare value required to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_DELAY_DITHER

Description:

- HELPER macro calculating the compare value, with dithering feature enabled, to achieve the required timer output compare active/inactive delay.

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)

Return value:

- Compare: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_DELAY_DITHER (1000000, LL_TIM_GetPrescaler (), 10);

__LL_TIM_CALC_PULSE

Description:

- HELPER macro calculating the auto-reload value to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_CALC_PULSE_DITHER

Description:

- HELPER macro calculating the auto-reload value, with dithering feature enabled, to achieve the required pulse duration (when the timer operates in one pulse mode).

Parameters:

- __TIMCLK__: timer input clock frequency (in Hz)
- __PSC__: prescaler
- __DELAY__: timer output compare active/inactive delay (in us)
- __PULSE__: pulse duration (in us)

Return value:

- Auto-reload: value (between Min_Data=0 and Max_Data=65535)

Notes:

- ex: __LL_TIM_CALC_PULSE_DITHER (1000000, LL_TIM_GetPrescaler (), 10, 20);

__LL_TIM_GET_ICPSC_RATIO

Description:

- HELPER macro retrieving the ratio of the input capture prescaler.

Parameters:

- __ICPSC__: This parameter can be one of the following values:
  - LL_TIM_ICPSC_DIV1
  - LL_TIM_ICPSC_DIV2
  - LL_TIM_ICPSC_DIV4
  - LL_TIM_ICPSC_DIV8

Return value:

- Input: capture prescaler ratio (1, 2, 4 or 8)

Notes:

- ex: __LL_TIM_GET_ICPSC_RATIO (LL_TIM_IC_GetPrescaler ());

120 LL UCPD Generic Driver

120.1 UCPD Firmware driver registers structures

120.1.1 LL_UCPD_InitTypeDef

LL_UCPD_InitTypeDef is defined in the stm32u5xx_ll_ucpd.h

Data Fields

- ***uint32_t psc_ucpdclk***
- ***uint32_t transwin***
- ***uint32_t IfrGap***
- ***uint32_t HbitClockDiv***

Field Documentation

- ***uint32_t LL_UCPD_InitTypeDef::psc_ucpdclk***
  Specify the prescaler for the UCPD clock. This parameter can be a value of ***UCPD_LL_EC_PSC***. This feature can be modified afterwards using function **LL_UCPD_SetPSCClk()**.
- ***uint32_t LL_UCPD_InitTypeDef::transwin***
  Specify the number of cycles (minus 1) of the half bit clock (see HBITCLKDIV) to achieve a legal tTransitionWindow (set according to peripheral clock to define an interval of between 12 and 20 us). This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This value can be modified afterwards using function **LL_UCPD_SetTransWin()**.
- ***uint32_t LL_UCPD_InitTypeDef::IfrGap***
  Specify the definition of the clock divider (minus 1) in order to generate tInterframeGap from the peripheral clock. This parameter can be a value between Min_Data=0x1 and Max_Data=0x1F This feature can be modified afterwards using function **LL_UCPD_SetIfrGap()**.
- ***uint32_t LL_UCPD_InitTypeDef::HbitClockDiv***
  Specify the number of cycles (minus one) at UCPD peripheral for a half bit clock e.g. program 3 for a bit clock that takes 8 cycles of the peripheral clock : "UCPD1_CLK". This parameter can be a value between Min_Data=0x0 and Max_Data=0x3F. This feature can be modified using function **LL_UCPD_SetHbitClockDiv()**.

120.2 UCPD Firmware driver API description

The following section lists the various functions of the UCPD library.

120.2.1 Detailed description of functions

LL_UCPD_Enable

Function name

__STATIC_INLINE void LL_UCPD_Enable (UCPD_TypeDef * UCPDx)

Function description

Enable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Enable

LL_UCPD_Disable

Function name

__STATIC_INLINE void LL_UCPD_Disable (UCPD_TypeDef * UCPDx)

Function description

Disable UCPD peripheral.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Notes

- When disabling the UCPD, follow the procedure described in the Reference Manual.

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_Disable

LL_UCPD_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabled (UCPD_TypeDef const *const UCPDx)

Function description

Check if UCPD peripheral is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFG1 UCPDEN LL_UCPD_IsEnabled

LL_UCPD_SetRxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrderSet (UCPD_TypeDef * UCPDx, uint32_t OrderSet)

Function description

Set the receiver ordered set detection enable.

Parameters

- **UCPDx:** UCPD Instance
- **OrderSet:** This parameter can be combination of the following values:
  – LL_UCPD_ORDERSET_SOP
  – LL_UCPD_ORDERSET_SOP1
  – LL_UCPD_ORDERSET_SOP2
  – LL_UCPD_ORDERSET_HARDRST
  – LL_UCPD_ORDERSET_CABLERST
  – LL_UCPD_ORDERSET_SOP1_DEBUG
  – LL_UCPD_ORDERSET_SOP2_DEBUG
  – LL_UCPD_ORDERSET_SOP_EXT1
  – LL_UCPD_ORDERSET_SOP_EXT2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXORDSETEN LL_UCPD_SetRxOrderSet

LL_UCPD_SetPSCClk

Function name

__STATIC_INLINE void LL_UCPD_SetPSCClk (UCPD_TypeDef * UCPDx, uint32_t Psc)

Function description

Set the prescaler for ucpd clock.

Parameters

- **UCPDx:** UCPD Instance
- **Psc:** This parameter can be one of the following values:
  – LL_UCPD_PSC_DIV1
  – LL_UCPD_PSC_DIV2
  – LL_UCPD_PSC_DIV4
  – LL_UCPD_PSC_DIV8
  – LL_UCPD_PSC_DIV16

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 UCPDCLK LL_UCPD_SetPSCClk

LL_UCPD_SetTransWin

Function name

__STATIC_INLINE void LL_UCPD_SetTransWin (UCPD_TypeDef * UCPDx, uint32_t TransWin)

Function description

Set the number of cycles (minus 1) of the half bit clock.

Parameters

- **UCPDx:** UCPD Instance
- **TransWin:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TRANSWIN LL_UCPD_SetTransWin

LL_UCPD_SetIfrGap

Function name

__STATIC_INLINE void LL_UCPD_SetIfrGap (UCPD_TypeDef * UCPDx, uint32_t IfrGap)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **IfrGap:** a value between Min_Data=0x1 and Max_Data=0x1F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 IFRGAP LL_UCPD_SetIfrGap

LL_UCPD_SetHbitClockDiv

Function name

__STATIC_INLINE void LL_UCPD_SetHbitClockDiv (UCPD_TypeDef * UCPDx, uint32_t HbitClock)

Function description

Set the clock divider value to generate an interframe gap.

Parameters

- **UCPDx:** UCPD Instance
- **HbitClock:** a value between Min_Data=0x0 and Max_Data=0x3F

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 HBITCLKDIV LL_UCPD_SetHbitClockDiv

LL_UCPD_RxAnalogFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxAnalogFilterEnable (UCPD_TypeDef * UCPDx)

Function description

Enable Rx Analog Filter.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXAFILTEN LL_UCPD_RxAnalogFilterEnable

LL_UCPD_RxAnalogFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxAnalogFilterDisable (UCPD_TypeDef * UCPDx)

Function description

Disable Rx Analog Filter.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXAFILTEN LL_UCPD_RxAnalogFilterDisable

LL_UCPD_WakeUpEnable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpEnable (UCPD_TypeDef * UCPDx)

Function description

Enable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpEnable

LL_UCPD_WakeUpDisable

Function name

__STATIC_INLINE void LL_UCPD_WakeUpDisable (UCPD_TypeDef * UCPDx)

Function description

Disable the wakeup mode.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 WUPEN LL_UCPD_WakeUpDisable

LL_UCPD_ForceClockEnable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockEnable (UCPD_TypeDef * UCPDx)

Function description

Force clock enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockEnable

LL_UCPD_ForceClockDisable

Function name

__STATIC_INLINE void LL_UCPD_ForceClockDisable (UCPD_TypeDef * UCPDx)

Function description

Force clock disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 FORCECLK LL_UCPD_ForceClockDisable

LL_UCPD_RxFilterEnable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterEnable (UCPD_TypeDef * UCPDx)

Function description

RxFilter enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterEnable

LL_UCPD_RxFilterDisable

Function name

__STATIC_INLINE void LL_UCPD_RxFilterDisable (UCPD_TypeDef * UCPDx)

Function description

RxFilter disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG2 RXFILTDIS LL_UCPD_RxFilterDisable

LL_UCPD_TypeCDetectionCC2Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Enable

LL_UCPD_TypeCDetectionCC2Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC2Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC2 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC2TCDIS LL_UCPD_TypeCDetectionCC2Disable

LL_UCPD_TypeCDetectionCC1Enable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Enable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Enable

LL_UCPD_TypeCDetectionCC1Disable

Function name

__STATIC_INLINE void LL_UCPD_TypeCDetectionCC1Disable (UCPD_TypeDef * UCPDx)

Function description

Type C detector for CC1 disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1TCDIS LL_UCPD_TypeCDetectionCC1Disable

LL_UCPD_VconnDischargeEnable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeEnable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeEnable

LL_UCPD_VconnDischargeDisable

Function name

__STATIC_INLINE void LL_UCPD_VconnDischargeDisable (UCPD_TypeDef * UCPDx)

Function description

Source Vconn discharge disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RDCH LL_UCPD_VconnDischargeDisable

LL_UCPD_SignalFRSTX

Function name

__STATIC_INLINE void LL_UCPD_SignalFRSTX (UCPD_TypeDef * UCPDx)

Function description

Signal Fast Role Swap request.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSTX LL_UCPD_VconnDischargeDisable

LL_UCPD_FRSDetectionEnable

Function name

__STATIC_INLINE void LL_UCPD_FRSDetectionEnable (UCPD_TypeDef * UCPDx)

Function description

Fast Role swap RX detection enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSRXEN LL_UCPD_FRSDetectionEnable

LL_UCPD_FRSDetectionDisable

Function name

__STATIC_INLINE void LL_UCPD_FRSDetectionDisable (UCPD_TypeDef * UCPDx)

Function description

Fast Role swap RX detection disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR FRSRXEN LL_UCPD_FRSDetectionDisable

LL_UCPD_SetccEnable

Function name

__STATIC_INLINE void LL_UCPD_SetccEnable (UCPD_TypeDef * UCPDx, uint32_t CCEnable)

Function description

Set cc enable.

Parameters

- **UCPDx:** UCPD Instance
- **CCEnable:** This parameter can be one of the following values:
  – LL_UCPD_CCENABLE_NONE
  – LL_UCPD_CCENABLE_CC1
  – LL_UCPD_CCENABLE_CC2
  – LL_UCPD_CCENABLE_CC1CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR CC1VCONNEN LL_UCPD_SetccEnable

LL_UCPD_SetSNKRole

Function name

__STATIC_INLINE void LL_UCPD_SetSNKRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SNK role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSNKRole

LL_UCPD_SetSRCRole

Function name

__STATIC_INLINE void LL_UCPD_SetSRCRole (UCPD_TypeDef * UCPDx)

Function description

Set UCPD SRC role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_SetSRCRole

LL_UCPD_GetRole

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetRole (UCPD_TypeDef const *const UCPDx)

Function description

Get UCPD Role.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_UCPD_ROLE_SNK
  - LL_UCPD_ROLE_SRC

Reference Manual to LL API cross reference:

- CR ANAMODE LL_UCPD_GetRole

LL_UCPD_SetRpResistor

Function name

__STATIC_INLINE void LL_UCPD_SetRpResistor (UCPD_TypeDef * UCPDx, uint32_t Resistor)

Function description

Set Rp resistor.

Parameters

- **UCPDx:** UCPD Instance
- **Resistor:** This parameter can be one of the following values:
  - LL_UCPD_RESISTOR_DEFAULT
  - LL_UCPD_RESISTOR_1_5A
  - LL_UCPD_RESISTOR_3_0A
  - LL_UCPD_RESISTOR_NONE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR ANASUBMODE LL_UCPD_SetRpResistor

LL_UCPD_SetCCPin

Function name

__STATIC_INLINE void LL_UCPD_SetCCPin (UCPD_TypeDef * UCPDx, uint32_t CCPin)

Function description

Set CC pin.

Parameters

- **UCPDx:** UCPD Instance
- **CCPin:** This parameter can be one of the following values:
  - LL_UCPD_CCPIN_CC1
  - LL_UCPD_CCPIN_CC2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYCCSEL LL_UCPD_SetCCPin

LL_UCPD_RxEnable

Function name

__STATIC_INLINE void LL_UCPD_RxEnable (UCPD_TypeDef * UCPDx)

Function description

Rx enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxEnable

LL_UCPD_RxDisable

Function name

__STATIC_INLINE void LL_UCPD_RxDisable (UCPD_TypeDef * UCPDx)

Function description

Rx disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR PHYRXEN LL_UCPD_RxDisable

LL_UCPD_SetRxMode

Function name

__STATIC_INLINE void LL_UCPD_SetRxMode (UCPD_TypeDef * UCPDx, uint32_t RxMode)

Function description

Set Rx mode.

Parameters

- **UCPDx:** UCPD Instance
- **RxMode:** This parameter can be one of the following values:
  – LL_UCPD_RXMODE_NORMAL
  – LL_UCPD_RXMODE_BIST_TEST_DATA

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR RXMODE LL_UCPD_SetRxMode

LL_UCPD_SendHardReset

Function name

__STATIC_INLINE void LL_UCPD_SendHardReset (UCPD_TypeDef * UCPDx)

Function description

Send Hard Reset.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXHRST LL_UCPD_SendHardReset

LL_UCPD_SendMessage

Function name

__STATIC_INLINE void LL_UCPD_SendMessage (UCPD_TypeDef * UCPDx)

Function description

Send message.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXSEND LL_UCPD_SendMessage

LL_UCPD_SetTxMode

Function name

__STATIC_INLINE void LL_UCPD_SetTxMode (UCPD_TypeDef * UCPDx, uint32_t TxMode)

Function description

Set Tx mode.

Parameters

- **UCPDx:** UCPD Instance
- **TxMode:** This parameter can be one of the following values:
  – LL_UCPD_TXMODE_NORMAL
  – LL_UCPD_TXMODE_CABLE_RESET
  – LL_UCPD_TXMODE_BIST_CARRIER2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR TXMODE LL_UCPD_SetTxMode

LL_UCPD_EnableIT_FRS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_FRS (UCPD_TypeDef * UCPDx)

Function description

Enable FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_EnableIT_FRS

LL_UCPD_EnableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_EnableIT_TypeCEventCC2

LL_UCPD_EnableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Enable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_EnableIT_TypeCEventCC1

LL_UCPD_EnableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Enable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_EnableIT_RxMsgEnd

LL_UCPD_EnableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Enable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_EnableIT_RxOvr

LL_UCPD_EnableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Enable Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_EnableIT_RxHRST

LL_UCPD_EnableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Enable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_EnableIT_RxOrderSet

LL_UCPD_EnableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Enable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_EnableIT_RxNE

LL_UCPD_EnableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Enable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_EnableIT_TxUND

LL_UCPD_EnableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_EnableIT_TxHRSTSENT

LL_UCPD_EnableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Enable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_EnableIT_TxHRSTDISC

LL_UCPD_EnableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_EnableIT_TxMSGABT

LL_UCPD_EnableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_EnableIT_TxMSGSENT

LL_UCPD_EnableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Enable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_EnableIT_TxMSGDISC

LL_UCPD_EnableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_EnableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Enable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_EnableIT_TxIS

LL_UCPD_DisableIT_FRS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_FRS (UCPD_TypeDef * UCPDx)

Function description

Disable FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_DisableIT_FRS

LL_UCPD_DisableIT_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_DisableIT_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Disable type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TYPECEVT1IE LL_UCPD_DisableIT_TypeCEventCC1

LL_UCPD_DisableIT_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Disable Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_DisableIT_RxMsgEnd

LL_UCPD_DisableIT_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Disable Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_DisableIT_RxOvr

LL_UCPD_DisableIT_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Disable Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_DisableIT_RxHRST

LL_UCPD_DisableIT_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Disable Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_DisableIT_RxOrderSet

LL_UCPD_DisableIT_RxNE

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_RxNE (UCPD_TypeDef * UCPDx)

Function description

Disable Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_DisableIT_RxNE

LL_UCPD_DisableIT_TxUND

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxUND (UCPD_TypeDef * UCPDx)

Function description

Disable TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_DisableIT_TxUND

LL_UCPD_DisableIT_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_DisableIT_TxHRSTSENT

LL_UCPD_DisableIT_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Disable hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_DisableIT_TxHRSTDISC

LL_UCPD_DisableIT_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_DisableIT_TxMSGABT

LL_UCPD_DisableIT_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_DisableIT_TxMSGSENT

LL_UCPD_DisableIT_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Disable Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_DisableIT_TxMSGDISC

LL_UCPD_DisableIT_TxIS

Function name

__STATIC_INLINE void LL_UCPD_DisableIT_TxIS (UCPD_TypeDef * UCPDx)

Function description

Disable Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_DisableIT_TxIS

LL_UCPD_IsEnableIT_FRS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_FRS (UCPD_TypeDef const *const UCPDx)

Function description

Check if FRS interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR FRSEVTIE LL_UCPD_DisableIT_FRS

LL_UCPD_IsEnableIT_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TYPECEVT2IE LL_UCPD_DisableIT_TypeCEventCC2

LL_UCPD_IsEnableIT_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1 enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR2 TYPECEVT1IE LL_UCPD_IsEnableIT_TypeCEventCC1

LL_UCPD_IsEnableIT_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXMSGENDIE LL_UCPD_IsEnableIT_RxMsgEnd

LL_UCPD_IsEnableIT_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXOVRIE LL_UCPD_IsEnableIT_RxOvr

LL_UCPD_IsEnableIT_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard resrt interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXHRSTDETIE LL_UCPD_IsEnableIT_RxHRST

LL_UCPD_IsEnableIT_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXORDDETIE LL_UCPD_IsEnableIT_RxOrderSet

LL_UCPD_IsEnableIT_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR RXNEIE LL_UCPD_IsEnableIT_RxNE

LL_UCPD_IsEnableIT_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXUNDIE LL_UCPD_IsEnableIT_TxUND

LL_UCPD_IsEnableIT_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTSENTIE LL_UCPD_IsEnableIT_TxHRSTSENT

LL_UCPD_IsEnableIT_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR HRSTDISCIE LL_UCPD_IsEnableIT_TxHRSTDISC

LL_UCPD_IsEnableIT_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGABTIE LL_UCPD_IsEnableIT_TxMSGABT

LL_UCPD_IsEnableIT_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGSENTIE LL_UCPD_IsEnableIT_TxMSGSENT

LL_UCPD_IsEnableIT_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXMSGDISCIE LL_UCPD_IsEnableIT_TxMSGDISC

LL_UCPD_IsEnableIT_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnableIT_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IMR TXISIE LL_UCPD_IsEnableIT_TxIS

LL_UCPD_ClearFlag_FRS

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_FRS (UCPD_TypeDef * UCPDx)

Function description

Clear FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FRSEVTIE LL_UCPD_ClearFlag_FRS

LL_UCPD_ClearFlag_TypeCEventCC2

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC2 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT2IE LL_UCPD_ClearFlag_TypeCEventCC2

LL_UCPD_ClearFlag_TypeCEventCC1

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TypeCEventCC1 (UCPD_TypeDef * UCPDx)

Function description

Clear type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IIMR TYPECEVT1IE LL_UCPD_ClearFlag_TypeCEventCC1

LL_UCPD_ClearFlag_RxMsgEnd

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxMsgEnd (UCPD_TypeDef * UCPDx)

Function description

Clear Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXMSGENDIE LL_UCPD_ClearFlag_RxMsgEnd

LL_UCPD_ClearFlag_RxOvr

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOvr (UCPD_TypeDef * UCPDx)

Function description

Clear Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXOVRIE LL_UCPD_ClearFlag_RxOvr

LL_UCPD_ClearFlag_RxHRST

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxHRST (UCPD_TypeDef * UCPDx)

Function description

Clear Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXHRSTDETIE LL_UCPD_ClearFlag_RxHRST

LL_UCPD_ClearFlag_RxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_RxOrderSet (UCPD_TypeDef * UCPDx)

Function description

Clear Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RXORDDETIE LL_UCPD_ClearFlag_RxOrderSet

LL_UCPD_ClearFlag_TxUND

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxUND (UCPD_TypeDef * UCPDx)

Function description

Clear TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXUNDIE LL_UCPD_ClearFlag_TxUND

LL_UCPD_ClearFlag_TxHRSTSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTSENT (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTSENTIE LL_UCPD_ClearFlag_TxHRSTSENT

LL_UCPD_ClearFlag_TxHRSTDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxHRSTDISC (UCPD_TypeDef * UCPDx)

Function description

Clear hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR HRSTDISCIE LL_UCPD_ClearFlag_TxHRSTDISC

LL_UCPD_ClearFlag_TxMSGABT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGABT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGABTIE LL_UCPD_ClearFlag_TxMSGABT

LL_UCPD_ClearFlag_TxMSGSENT

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGSENT (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGSENTIE LL_UCPD_ClearFlag_TxMSGSENT

LL_UCPD_ClearFlag_TxMSGDISC

Function name

__STATIC_INLINE void LL_UCPD_ClearFlag_TxMSGDISC (UCPD_TypeDef * UCPDx)

Function description

Clear Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TXMSGDISCIE LL_UCPD_ClearFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_FRS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_FRS (UCPD_TypeDef const *const UCPDx)

Function description

Check if FRS interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR FRSEVT LL_UCPD_IsActiveFlag_FRS

LL_UCPD_IsActiveFlag_TypeCEventCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC2 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC2.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TYPECEVT2 LL_UCPD_IsActiveFlag_TypeCEventCC2

LL_UCPD_IsActiveFlag_TypeCEventCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TypeCEventCC1 (UCPD_TypeDef const *const UCPDx)

Function description

Check if type c event on CC1.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR TYPECEVT1 LL_UCPD_IsActiveFlag_TypeCEventCC1

LL_UCPD_IsActiveFlag_RxMsgEnd

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxMsgEnd (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx message end interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXMSGEND LL_UCPD_IsActiveFlag_RxMsgEnd

LL_UCPD_IsActiveFlag_RxOvr

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOvr (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx overrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXOVR LL_UCPD_IsActiveFlag_RxOvr

LL_UCPD_IsActiveFlag_RxHRST

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxHRST (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx hard resrt interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR RXHRSTDET LL_UCPD_IsActiveFlag_RxHRST

LL_UCPD_IsActiveFlag_RxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx orderset interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXORDDET LL_UCPD_IsActiveFlag_RxOrderSet

LL_UCPD_IsActiveFlag_RxNE

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_RxNE (UCPD_TypeDef const *const UCPDx)

Function description

Check if Rx non empty interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_UCPD_IsActiveFlag_RxNE

LL_UCPD_IsActiveFlag_TxUND

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxUND (UCPD_TypeDef const *const UCPDx)

Function description

Check if TX underrun interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXUND LL_UCPD_IsActiveFlag_TxUND

LL_UCPD_IsActiveFlag_TxHRSTSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTSENT LL_UCPD_IsActiveFlag_TxHRSTSENT

LL_UCPD_IsActiveFlag_TxHRSTDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxHRSTDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if hard reset discard interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR HRSTDISC LL_UCPD_IsActiveFlag_TxHRSTDISC

LL_UCPD_IsActiveFlag_TxMSGABT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGABT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message abort interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGABT LL_UCPD_IsActiveFlag_TxMSGABT

LL_UCPD_IsActiveFlag_TxMSGSENT

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGSENT (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message sent interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGSENT LL_UCPD_IsActiveFlag_TxMSGSENT

LL_UCPD_IsActiveFlag_TxMSGDISC

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxMSGDISC (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx message discarded interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXMSGDISC LL_UCPD_IsActiveFlag_TxMSGDISC

LL_UCPD_IsActiveFlag_TxIS

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsActiveFlag_TxIS (UCPD_TypeDef const *const UCPDx)

Function description

Check if Tx data receive interrupt.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_IsActiveFlag_TxIS

LL_UCPD_GetTypeCVstateCC2

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC2 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC2

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC2

LL_UCPD_GetTypeCVstateCC1

Function name

__STATIC_INLINE uint32_t LL_UCPD_GetTypeCVstateCC1 (UCPD_TypeDef const *const UCPDx)

Function description

return the vstate value for CC1

Parameters

- **UCPDx:** UCPD Instance

Return values

- **val:**

Reference Manual to LL API cross reference:

- SR TXIS LL_UCPD_GetTypeCVstateCC1

LL_UCPD_RxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_RxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMAEnable

LL_UCPD_RxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_RxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Rx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 RXDMAEN LL_UCPD_RxDMADisable

LL_UCPD_TxDMAEnable

Function name

__STATIC_INLINE void LL_UCPD_TxDMAEnable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Enable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMAEnable

LL_UCPD_TxDMADisable

Function name

__STATIC_INLINE void LL_UCPD_TxDMADisable (UCPD_TypeDef * UCPDx)

Function description

Tx DMA Disable.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFG1 TXDMAEN LL_UCPD_TxDMADisable

LL_UCPD_IsEnabledTxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledTxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Tx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_UCPD_IsEnabledTxDMA

LL_UCPD_IsEnabledRxDMA

Function name

__STATIC_INLINE uint32_t LL_UCPD_IsEnabledRxDMA (UCPD_TypeDef const *const UCPDx)

Function description

Check if DMA Rx is enabled.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_UCPD_IsEnabledRxDMA

LL_UCPD_WriteTxOrderSet

Function name

__STATIC_INLINE void LL_UCPD_WriteTxOrderSet (UCPD_TypeDef * UCPDx, uint32_t TxOrderSet)

Function description

write the orderset for Tx message

Parameters

- **UCPDx:** UCPD Instance
- **TxOrderSet:** one of the following value
  - LL_UCPD_ORDERED_SET_SOP
  - LL_UCPD_ORDERED_SET_SOP1
  - LL_UCPD_ORDERED_SET_SOP2
  - LL_UCPD_ORDERED_SET_HARD_RESET
  - LL_UCPD_ORDERED_SET_CABLE_RESET
  - LL_UCPD_ORDERED_SET_SOP1_DEBUG
  - LL_UCPD_ORDERED_SET_SOP2_DEBUG

Return values

- **None:**

Reference Manual to LL API cross reference:

- TX_ORDSET TXORDSET LL_UCPD_WriteTxOrderSet

LL_UCPD_WriteTxPaySize

Function name

__STATIC_INLINE void LL_UCPD_WriteTxPaySize (UCPD_TypeDef * UCPDx, uint32_t TxPaySize)

Function description

write the Tx paysize

Parameters

- **UCPDx:** UCPD Instance
- **TxPaySize:**

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TX_PAYSZ TXPAYSZ LL_UCPD_WriteTxPaySize

LL_UCPD_WriteData

Function name

__STATIC_INLINE void LL_UCPD_WriteData (UCPD_TypeDef * UCPDx, uint8_t Data)

Function description

Write data.

Parameters

- **UCPDx:** UCPD Instance
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None.:**

Reference Manual to LL API cross reference:

- TXDR DR LL_UCPD_WriteData

LL_UCPD_ReadRxOrderSet

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxOrderSet (UCPD_TypeDef const *const UCPDx)

Function description

read RX the orderset

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxOrderSet:** one of the following value
  - LL_UCPD_RXORDSET_SOP
  - LL_UCPD_RXORDSET_SOP1
  - LL_UCPD_RXORDSET_SOP2
  - LL_UCPD_RXORDSET_SOP1_DEBUG
  - LL_UCPD_RXORDSET_SOP2_DEBUG
  - LL_UCPD_RXORDSET_CABLE_RESET
  - LL_UCPD_RXORDSET_SOPEXT1
  - LL_UCPD_RXORDSET_SOPEXT2

Reference Manual to LL API cross reference:

- RX_ORDSET RXORDSET LL_UCPD_ReadRxOrderSet

LL_UCPD_ReadRxPaySize

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadRxPaySize (UCPD_TypeDef const *const UCPDx)

Function description

Read the Rx paysize.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RXPaysize.:**

Reference Manual to LL API cross reference:

- RX_PAYSZ RXPAYSZ LL_UCPD_ReadRxPaySize

LL_UCPD_ReadData

Function name

__STATIC_INLINE uint32_t LL_UCPD_ReadData (UCPD_TypeDef const *const UCPDx)

Function description

Read data.

Parameters

- **UCPDx:** UCPD Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- TXDR RXDATA LL_UCPD_ReadData

LL_UCPD_SetRxOrdExt1

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt1 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext1.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT1 RXSOPX1 LL_UCPD_SetRxOrdExt1

LL_UCPD_SetRxOrdExt2

Function name

__STATIC_INLINE void LL_UCPD_SetRxOrdExt2 (UCPD_TypeDef * UCPDx, uint32_t SOPExt)

Function description

Set Rx OrderSet Ext2.

Parameters

- **UCPDx:** UCPD Instance
- **SOPExt:** Value between Min_Data=0x00000 and Max_Data=0xFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RX_ORDEXT2 RXSOPX2 LL_UCPD_SetRxOrdExt2

LL_UCPD_DeInit

Function name

ErrorStatus LL_UCPD_DeInit (UCPD_TypeDef * UCPDx)

Function description

De-initialize the UCPD registers to their default reset values.

Parameters

- **UCPDx:** ucpd Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: ucpd registers are de-initialized
  - ERROR: ucpd registers are not de-initialized

LL_UCPD_Init

Function name

ErrorStatus LL_UCPD_Init (UCPD_TypeDef * UCPDx, LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Initialize the ucpd registers according to the specified parameters in UCPD_InitStruct.

Parameters

- **UCPDx:** UCPD Instance
- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure that contains the configuration information for the UCPD peripheral.

Return values

- **An:** ErrorStatus enumeration value. (Return always SUCCESS)

Notes

- As some bits in ucpd configuration registers can only be written when the ucpd is disabled (ucpd_CR1_SPE bit =0), UCPD peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_UCPD_StructInit

Function name

void LL_UCPD_StructInit (LL_UCPD_InitTypeDef * UCPD_InitStruct)

Function description

Set each LL_UCPD_InitTypeDef field to default value.

Parameters

- **UCPD_InitStruct:** pointer to a LL_UCPD_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

120.3 UCPD Firmware driver defines

The following section lists the various define and macros of the module.

120.3.1 UCPD

UCPD

CC pin enable

LL_UCPD_CCENABLE_NONE

Neither PHY is activated (e.g. disabled state of source)

LL_UCPD_CCENABLE_CC1

Controls apply to only CC1

LL_UCPD_CCENABLE_CC2

Controls apply to only CC1

LL_UCPD_CCENABLE_CC1CC2

Controls apply to both CC1 and CC2 (normal usage for sink/source)

CC pin selection

LL_UCPD_CCPIN_CC1

Use CC1 IO for power delivery communication

LL_UCPD_CCPIN_CC2

Use CC2 IO for power delivery communication

CCx event

LL_UCPD_SNK_CC1_VOPEN

CC1 Sink Open state

LL_UCPD_SNK_CC1_VRP

CC1 Sink vRP default state

LL_UCPD_SNK_CC1_VRP15A

CC1 Sink vRP 1.5A state

LL_UCPD_SNK_CC1_VRP30A

CC1 Sink vRP 3.0A state

LL_UCPD_SNK_CC2_VOPEN

CC2 Sink Open state

LL_UCPD_SNK_CC2_VRP

CC2 Sink vRP default state

LL_UCPD_SNK_CC2_VRP15A

CC2 Sink vRP 1.5A state

LL_UCPD_SNK_CC2_VRP30A

CC2 Sink vRP 3.0A state

LL_UCPD_SRC_CC1_VRA

CC1 Source vRA state

LL_UCPD_SRC_CC1_VRD

CC1 Source vRD state

LL_UCPD_SRC_CC1_OPEN

CC1 Source Open state

LL_UCPD_SRC_CC2_VRA

CC2 Source vRA state

LL_UCPD_SRC_CC2_VRD

CC2 Source vRD state

LL_UCPD_SRC_CC2_OPEN

CC2 Source Open state

ordered set configuration

LL_UCPD_ORDERSET_SOP

SOP Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1

SOP' Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2

SOP'' Ordered set detection enabled

LL_UCPD_ORDERSET_HARDRST

Hard Reset Ordered set detection enabled

LL_UCPD_ORDERSET_CABLERST

Cable Reset Ordered set detection enabled

LL_UCPD_ORDERSET_SOP1_DEBUG

SOP' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP2_DEBUG

SOP'' Debug Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT1

SOP extension#1 Ordered set detection enabled

LL_UCPD_ORDERSET_SOP_EXT2

SOP extension#2 Ordered set detection enabled

Get Flags Defines

LL_UCPD_SR_TXIS

Transmit interrupt status

LL_UCPD_SR_TXMSGDISC

Transmit message discarded interrupt

LL_UCPD_SR_TXMSGSENT

Transmit message sent interrupt

LL_UCPD_SR_TXMSGABT

Transmit message abort interrupt

LL_UCPD_SR_HRSTDISC

HRST discarded interrupt

LL_UCPD_SR_HRSTSENT

HRST sent interrupt

LL_UCPD_SR_TXUND

Tx data underrun condition interrupt

LL_UCPD_SR_RXNE

Receive data register not empty interrupt

LL_UCPD_SR_RXORDDET

Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_SR_RXHRSTDET

Rx Hard Reset detect interrupt

LL_UCPD_SR_RXOVR

Rx data overflow interrupt

LL_UCPD_SR_RXMSGEND

Rx message received

LL_UCPD_SR_RXERR

Rx error

LL_UCPD_SR_TYPECEVT1

Type C voltage level event on CC1

LL_UCPD_SR_TYPECEVT2

Type C voltage level event on CC2

LL_UCPD_SR_TYPEC_VSTATE_CC1

Status of DC level on CC1 pin

LL_UCPD_SR_TYPEC_VSTATE_CC2

Status of DC level on CC2 pin

LL_UCPD_SR_FRSEVT

Fast Role Swap detection event

IT Defines

LL_UCPD_IMR_TXIS

Enable transmit interrupt status

LL_UCPD_IMR_TXMSGDISC

Enable transmit message discarded interrupt

LL_UCPD_IMR_TXMSGSENT

Enable transmit message sent interrupt

LL_UCPD_IMR_TXMSGABT

Enable transmit message abort interrupt

LL_UCPD_IMR_HRSTDISC

Enable HRST discarded interrupt

LL_UCPD_IMR_HRSTSENT

Enable HRST sent interrupt

LL_UCPD_IMR_TXUND

Enable tx data underrun condition interrupt

LL_UCPD_IMR_RXNE

Enable Receive data register not empty interrupt

LL_UCPD_IMR_RXORDDET

Enable Rx ordered set (4 K-codes) detected interrupt

LL_UCPD_IMR_RXHRSTDET

Enable Rx Hard Reset detect interrupt

LL_UCPD_IMR_RXOVR

Enable Rx data overflow interrupt

LL_UCPD_IMR_RXMSGEND

Enable Rx message received

LL_UCPD_IMR_TYPECEVT1

Enable Type C voltage level event on CC1

LL_UCPD_IMR_TYPECEVT2

Enable Type C voltage level event on CC2

LL_UCPD_IMR_FRSEVT

Enable fast Role Swap detection event

Role Mode

LL_UCPD_ROLE_SNK

Mode SNK Rd

LL_UCPD_ROLE_SRC

Mode SRC Rp

Ordered sets value

LL_UCPD_SYNC1

K-code for Startsynch #1

LL_UCPD_SYNC2

K-code for Startsynch #2

LL_UCPD_SYNC3

K-code for Startsynch #3

LL_UCPD_RST1

K-code for Hard Reset #1

LL_UCPD_RST2

K-code for Hard Reset #2

LL_UCPD_EOP

K-code for EOP End of Packet

LL_UCPD_ORDERED_SET_SOP

SOP Ordered set coding

LL_UCPD_ORDERED_SET_SOP1

SOP' Ordered set coding

LL_UCPD_ORDERED_SET_SOP2

SOP'' Ordered set coding

LL_UCPD_ORDERED_SET_HARD_RESET

Hard Reset Ordered set coding

LL_UCPD_ORDERED_SET_CABLE_RESET

Cable Reset Ordered set coding

LL_UCPD_ORDERED_SET_SOP1_DEBUG

SOP' Debug Ordered set coding

LL_UCPD_ORDERED_SET_SOP2_DEBUG

SOP'' Debug Ordered set coding

prescaler for UCPDCLK

LL_UCPD_PSC_DIV1

Bypass pre-scaling / divide by 1

LL_UCPD_PSC_DIV2

Pre-scale clock by dividing by 2

LL_UCPD_PSC_DIV4

Pre-scale clock by dividing by 4

LL_UCPD_PSC_DIV8

Pre-scale clock by dividing by 8

LL_UCPD_PSC_DIV16

Pre-scale clock by dividing by 16

Resistor value

LL_UCPD_RESISTOR_DEFAULT

Rp default

LL_UCPD_RESISTOR_1_5A

Rp 1.5 A

LL_UCPD_RESISTOR_3_0A

Rp 3.0 A

LL_UCPD_RESISTOR_NONE

No resistor

Receiver mode

LL_UCPD_RXMODE_NORMAL

Normal receive mode

LL_UCPD_RXMODE_BIST_TEST_DATA

BIST receive mode (BIST Test Data Mode)

Rx ordered set code detected

LL_UCPD_RXORDSET_SOP

SOP code detected in receiver

LL_UCPD_RXORDSET_SOP1

SOP' code detected in receiver

LL_UCPD_RXORDSET_SOP2

SOP'' code detected in receiver

LL_UCPD_RXORDSET_SOP1_DEBUG

SOP' Debug code detected in receiver

LL_UCPD_RXORDSET_SOP2_DEBUG

SOP'' Debug code detected in receiver

LL_UCPD_RXORDSET_CABLE_RESET

Cable Reset code detected in receiver

LL_UCPD_RXORDSET_SOPEXT1

SOP extension#1 code detected in receiver

LL_UCPD_RXORDSET_SOPEXT2

SOP extension#2 code detected in receiver

Type of Tx packet

LL_UCPD_TXMODE_NORMAL

Initiate the transfer of a Tx message

LL_UCPD_TXMODE_CABLE_RESET

Trigger a the transfer of a Cable Reset sequence

LL_UCPD_TXMODE_BIST_CARRIER2

Trigger a BIST test sequence send (BIST Carrier Mode 2)

Common Write and read registers Macros

LL_UCPD_WriteReg

Description:

- Write a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_UCPD_ReadReg

Description:

- Read a value in UCPD register.

Parameters:

- __INSTANCE__: UCPD Instance
- __REG__: Register to be read

Return value:

- Register: value

121 LL USART Generic Driver

121.1 USART Firmware driver registers structures

121.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32u5xx_ll_usart.h

Data Fields

- ***uint32_t PrescalerValue***
- ***uint32_t BaudRate***
- ***uint32_t DataWidth***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t TransferDirection***
- ***uint32_t HardwareFlowControl***
- ***uint32_t OverSampling***

Field Documentation

- ***uint32_t LL_USART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of ***USART_LL_EC_PRESCALER***.This feature can be modified afterwards using unitary function **LL_USART_SetPrescaler()**.
- ***uint32_t LL_USART_InitTypeDef::BaudRate***
  This field defines expected Usart communication baud rate.This feature can be modified afterwards using unitary function **LL_USART_SetBaudRate()**.
- ***uint32_t LL_USART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***USART_LL_EC_DATAWIDTH***.This feature can be modified afterwards using unitary function **LL_USART_SetDataWidth()**.
- ***uint32_t LL_USART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***USART_LL_EC_STOPBITS***.This feature can be modified afterwards using unitary function **LL_USART_SetStopBitsLength()**.
- ***uint32_t LL_USART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***USART_LL_EC_PARITY***.This feature can be modified afterwards using unitary function **LL_USART_SetParity()**.
- ***uint32_t LL_USART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_DIRECTION***.This feature can be modified afterwards using unitary function **LL_USART_SetTransferDirection()**.
- ***uint32_t LL_USART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***USART_LL_EC_HWCONTROL***.This feature can be modified afterwards using unitary function **LL_USART_SetHWFlowCtrl()**.
- ***uint32_t LL_USART_InitTypeDef::OverSampling***
  Specifies whether USART oversampling mode is 16 or 8. This parameter can be a value of ***USART_LL_EC_OVERSAMPLING***.This feature can be modified afterwards using unitary function **LL_USART_SetOverSampling()**.

121.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32u5xx_ll_usart.h

Data Fields

- ***uint32_t ClockOutput***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPhase***

- *uint32_t LastBitClockPulse*

Field Documentation

- ***uint32_t LL_USART_ClockInitTypeDef::ClockOutput***
Specifies whether the USART clock is enabled or disabled. This parameter can be a value of *USART_LL_EC_CLOCK*.USART HW configuration can be modified afterwards using unitary functions **LL_USART_EnableSCLKOutput()** or **LL_USART_DisableSCLKOutput()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPolarity***
Specifies the steady state of the serial clock. This parameter can be a value of *USART_LL_EC_POLARITY*.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPolarity()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::ClockPhase***
Specifies the clock transition on which the bit capture is made. This parameter can be a value of *USART_LL_EC_PHASE*.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetClockPhase()**. For more details, refer to description of this function.
- ***uint32_t LL_USART_ClockInitTypeDef::LastBitClockPulse***
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of *USART_LL_EC_LASTCLKPULSE*.USART HW configuration can be modified afterwards using unitary functions **LL_USART_SetLastClkPulseOutput()**. For more details, refer to description of this function.

121.2 USART Firmware driver API description

The following section lists the various functions of the USART library.

121.2.1 Detailed description of functions

LL_USART_Enable

Function name

__STATIC_INLINE void LL_USART_Enable (USART_TypeDef * USARTx)

Function description

USART Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Enable

LL_USART_Disable

Function name

__STATIC_INLINE void LL_USART_Disable (USART_TypeDef * USARTx)

Function description

USART Disable (all USART prescalers and outputs are disabled)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When USART is disabled, USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USARTx_ISR are set to their default values.

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_Disable

LL_USART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 UE LL_USART_IsEnabled

LL_USART_EnableFIFO

Function name

__STATIC_INLINE void LL_USART_EnableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_EnableFIFO

LL_USART_DisableFIFO

Function name

__STATIC_INLINE void LL_USART_DisableFIFO (USART_TypeDef * USARTx)

Function description

FIFO Mode Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_DisableFIFO

LL_USART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledFIFO (const USART_TypeDef * USARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_USART_IsEnabledFIFO

LL_USART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetTXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  – LL_USART_FIFOTHRESHOLD_1_8
  – LL_USART_FIFOTHRESHOLD_1_4
  – LL_USART_FIFOTHRESHOLD_1_2
  – LL_USART_FIFOTHRESHOLD_3_4
  – LL_USART_FIFOTHRESHOLD_7_8
  – LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_SetTXFIFOThreshold

LL_USART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_GetTXFIFOThreshold

LL_USART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_USART_SetRXFIFOThreshold (USART_TypeDef * USARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **USARTx:** USART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_SetRXFIFOThreshold

LL_USART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXFIFOThreshold (const USART_TypeDef * USARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_USART_GetRXFIFOThreshold

LL_USART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_USART_ConfigFIFOsThreshold (USART_TypeDef * USARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **USARTx:** USART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_USART_FIFOTHRESHOLD_1_8
  - LL_USART_FIFOTHRESHOLD_1_4
  - LL_USART_FIFOTHRESHOLD_1_2
  - LL_USART_FIFOTHRESHOLD_3_4
  - LL_USART_FIFOTHRESHOLD_7_8
  - LL_USART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_USART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_USART_ConfigFIFOsThreshold

LL_USART_EnableInStopMode

Function name

__STATIC_INLINE void LL_USART_EnableInStopMode (USART_TypeDef * USARTx)

Function description

USART enabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is enabled, USART is able to wake up the MCU from Stop mode, provided that USART clock selection is HSI or LSE in RCC.
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_EnableInStopMode

LL_USART_DisableInStopMode

Function name

__STATIC_INLINE void LL_USART_DisableInStopMode (USART_TypeDef * USARTx)

Function description

USART disabled in STOP Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When this function is disabled, USART is not able to wake up the MCU from Stop mode
- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_DisableInStopMode

LL_USART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledInStopMode (const USART_TypeDef * USARTx)

Function description

Indicate if USART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_WAKEUP_FROMSTOP_INSTANCE(USARTx) can be used to check whether or not Wake-up from Stop mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 UESM LL_USART_IsEnabledInStopMode

LL_USART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_EnableDirectionRx

LL_USART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionRx (USART_TypeDef * USARTx)

Function description

Receiver Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_DisableDirectionRx

LL_USART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_USART_EnableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Enable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_EnableDirectionTx

LL_USART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_USART_DisableDirectionTx (USART_TypeDef * USARTx)

Function description

Transmitter Disable.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_USART_DisableDirectionTx

LL_USART_SetTransferDirection

Function name

__STATIC_INLINE void LL_USART_SetTransferDirection (USART_TypeDef * USARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_SetTransferDirection
- CR1 TE LL_USART_SetTransferDirection

LL_USART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferDirection (const USART_TypeDef * USARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DIRECTION_NONE
  - LL_USART_DIRECTION_RX
  - LL_USART_DIRECTION_TX
  - LL_USART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_USART_GetTransferDirection
- CR1 TE LL_USART_GetTransferDirection

LL_USART_SetParity

Function name

__STATIC_INLINE void LL_USART_SetParity (USART_TypeDef * USARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled).

Parameters

- **USARTx:** USART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (9th or 8th bit depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_SetParity
- CR1 PCE LL_USART_SetParity

LL_USART_GetParity

Function name

__STATIC_INLINE uint32_t LL_USART_GetParity (const USART_TypeDef * USARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PARITY_NONE
  - LL_USART_PARITY_EVEN
  - LL_USART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_GetParity
- CR1 PCE LL_USART_GetParity

LL_USART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_USART_SetWakeUpMethod (USART_TypeDef * USARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance
- **Method:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_SetWakeUpMethod

LL_USART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_USART_GetWakeUpMethod (const USART_TypeDef * USARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_IDLELINE
  - LL_USART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_USART_GetWakeUpMethod

LL_USART_SetDataWidth

Function name

__STATIC_INLINE void LL_USART_SetDataWidth (USART_TypeDef * USARTx, uint32_t DataWidth)

Function description

Set Word length (i.e.

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_SetDataWidth
- CR1 M1 LL_USART_SetDataWidth

LL_USART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_USART_GetDataWidth (const USART_TypeDef * USARTx)

Function description

Return Word length (i.e.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_DATAWIDTH_7B
  - LL_USART_DATAWIDTH_8B
  - LL_USART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M0 LL_USART_GetDataWidth
- CR1 M1 LL_USART_GetDataWidth

LL_USART_EnableMuteMode

Function name

__STATIC_INLINE void LL_USART_EnableMuteMode (USART_TypeDef * USARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_EnableMuteMode

LL_USART_DisableMuteMode

Function name

__STATIC_INLINE void LL_USART_DisableMuteMode (USART_TypeDef * USARTx)

Function description

Prevent Mute Mode use.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_DisableMuteMode

LL_USART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledMuteMode (const USART_TypeDef * USARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 MME LL_USART_IsEnabledMuteMode

LL_USART_SetOverSampling

Function name

__STATIC_INLINE void LL_USART_SetOverSampling (USART_TypeDef * USARTx, uint32_t OverSampling)

Function description

Set Oversampling to 8-bit or 16-bit mode.

Parameters

- **USARTx:** USART Instance
- **OverSampling:** This parameter can be one of the following values:
  – LL_USART_OVERSAMPLING_16
  – LL_USART_OVERSAMPLING_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_SetOverSampling

LL_USART_GetOverSampling

Function name

__STATIC_INLINE uint32_t LL_USART_GetOverSampling (const USART_TypeDef * USARTx)

Function description

Return Oversampling mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_OVERSAMPLING_16
  – LL_USART_OVERSAMPLING_8

Reference Manual to LL API cross reference:

- CR1 OVER8 LL_USART_GetOverSampling

LL_USART_SetLastClkPulseOutput

Function name

__STATIC_INLINE void LL_USART_SetLastClkPulseOutput (USART_TypeDef * USARTx, uint32_t LastBitClockPulse)

Function description

Configure if Clock pulse of the last data bit is output to the SCLK pin or not.

Parameters

- **USARTx:** USART Instance
- **LastBitClockPulse:** This parameter can be one of the following values:
  – LL_USART_LASTCLKPULSE_NO_OUTPUT
  – LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_SetLastClkPulseOutput

LL_USART_GetLastClkPulseOutput

Function name

__STATIC_INLINE uint32_t LL_USART_GetLastClkPulseOutput (const USART_TypeDef * USARTx)

Function description

Retrieve Clock pulse of the last data bit output configuration (Last bit Clock pulse output to the SCLK pin or not)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_LASTCLKPULSE_NO_OUTPUT
  – LL_USART_LASTCLKPULSE_OUTPUT

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBCL LL_USART_GetLastClkPulseOutput

LL_USART_SetClockPhase

Function name

__STATIC_INLINE void LL_USART_SetClockPhase (USART_TypeDef * USARTx, uint32_t ClockPhase)

Function description

Select the phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPhase:** This parameter can be one of the following values:
  – LL_USART_PHASE_1EDGE
  – LL_USART_PHASE_2EDGE

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_SetClockPhase

LL_USART_GetClockPhase

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPhase (const USART_TypeDef * USARTx)

Function description

Return phase of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_GetClockPhase

LL_USART_SetClockPolarity

Function name

__STATIC_INLINE void LL_USART_SetClockPolarity (USART_TypeDef * USARTx, uint32_t ClockPolarity)

Function description

Select the polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_SetClockPolarity

LL_USART_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetClockPolarity (const USART_TypeDef * USARTx)

Function description

Return polarity of the clock output on the SCLK pin in synchronous mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CPOL LL_USART_GetClockPolarity

LL_USART_ConfigClock

Function name

__STATIC_INLINE void LL_USART_ConfigClock (USART_TypeDef * USARTx, uint32_t Phase, uint32_t Polarity, uint32_t LBCPOutput)

Function description

Configure Clock signal format (Phase Polarity and choice about output of last bit clock pulse)

Parameters

- **USARTx:** USART Instance
- **Phase:** This parameter can be one of the following values:
  - LL_USART_PHASE_1EDGE
  - LL_USART_PHASE_2EDGE
- **Polarity:** This parameter can be one of the following values:
  - LL_USART_POLARITY_LOW
  - LL_USART_POLARITY_HIGH
- **LBCPOutput:** This parameter can be one of the following values:
  - LL_USART_LASTCLKPULSE_NO_OUTPUT
  - LL_USART_LASTCLKPULSE_OUTPUT

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clock Phase configuration using LL_USART_SetClockPhase() functionClock Polarity configuration using LL_USART_SetClockPolarity() functionOutput of Last bit Clock pulse configuration using LL_USART_SetLastClkPulseOutput() function

Reference Manual to LL API cross reference:

- CR2 CPHA LL_USART_ConfigClock
- CR2 CPOL LL_USART_ConfigClock
- CR2 LBCL LL_USART_ConfigClock

LL_USART_SetPrescaler

Function name

__STATIC_INLINE void LL_USART_SetPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  – LL_USART_PRESCALER_DIV1
  – LL_USART_PRESCALER_DIV2
  – LL_USART_PRESCALER_DIV4
  – LL_USART_PRESCALER_DIV6
  – LL_USART_PRESCALER_DIV8
  – LL_USART_PRESCALER_DIV10
  – LL_USART_PRESCALER_DIV12
  – LL_USART_PRESCALER_DIV16
  – LL_USART_PRESCALER_DIV32
  – LL_USART_PRESCALER_DIV64
  – LL_USART_PRESCALER_DIV128
  – LL_USART_PRESCALER_DIV256

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_SetPrescaler

LL_USART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetPrescaler (const USART_TypeDef * USARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_USART_GetPrescaler

LL_USART_EnableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_EnableSCLKOutput (USART_TypeDef * USARTx)

Function description

Enable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_EnableSCLKOutput

LL_USART_DisableSCLKOutput

Function name

__STATIC_INLINE void LL_USART_DisableSCLKOutput (USART_TypeDef * USARTx)

Function description

Disable Clock output on SCLK pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_DisableSCLKOutput

LL_USART_IsEnabledSCLKOutput

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSCLKOutput (const USART_TypeDef * USARTx)

Function description

Indicate if Clock output on SCLK pin is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_IsEnabledSCLKOutput

LL_USART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_USART_SetStopBitsLength (USART_TypeDef * USARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **USARTx:** USART Instance
- **StopBits:** This parameter can be one of the following values:
  – LL_USART_STOPBITS_0_5
  – LL_USART_STOPBITS_1
  – LL_USART_STOPBITS_1_5
  – LL_USART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_SetStopBitsLength

LL_USART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetStopBitsLength (const USART_TypeDef * USARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_STOPBITS_0_5
  – LL_USART_STOPBITS_1
  – LL_USART_STOPBITS_1_5
  – LL_USART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_USART_GetStopBitsLength

LL_USART_ConfigCharacter

Function name

__STATIC_INLINE void LL_USART_ConfigCharacter (USART_TypeDef * USARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **USARTx:** USART Instance
- **DataWidth:** This parameter can be one of the following values:
  – LL_USART_DATAWIDTH_7B
  – LL_USART_DATAWIDTH_8B
  – LL_USART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  – LL_USART_PARITY_NONE
  – LL_USART_PARITY_EVEN
  – LL_USART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  – LL_USART_STOPBITS_0_5
  – LL_USART_STOPBITS_1
  – LL_USART_STOPBITS_1_5
  – LL_USART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_USART_SetDataWidth() functionParity Control and mode configuration using LL_USART_SetParity() functionStop bits configuration using LL_USART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_USART_ConfigCharacter
- CR1 PCE LL_USART_ConfigCharacter
- CR1 M0 LL_USART_ConfigCharacter
- CR1 M1 LL_USART_ConfigCharacter
- CR2 STOP LL_USART_ConfigCharacter

LL_USART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_USART_SetTXRXSwap (USART_TypeDef * USARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **USARTx:** USART Instance
- **SwapConfig:** This parameter can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_SetTXRXSwap

LL_USART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXRXSwap (const USART_TypeDef * USARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_TXRX_STANDARD
  - LL_USART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_USART_GetTXRXSwap

LL_USART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetRXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_SetRXPinLevel

LL_USART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetRXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_RXPIN_LEVEL_STANDARD
  - LL_USART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_USART_GetRXPinLevel

LL_USART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_USART_SetTXPinLevel (USART_TypeDef * USARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **USARTx:** USART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  - LL_USART_TXPIN_LEVEL_STANDARD
  - LL_USART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_SetTXPinLevel

LL_USART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_USART_GetTXPinLevel (const USART_TypeDef * USARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_TXPIN_LEVEL_STANDARD
  – LL_USART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_USART_GetTXPinLevel

LL_USART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_USART_SetBinaryDataLogic (USART_TypeDef * USARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **USARTx:** USART Instance
- **DataLogic:** This parameter can be one of the following values:
  – LL_USART_BINARY_LOGIC_POSITIVE
  – LL_USART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_SetBinaryDataLogic

LL_USART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_USART_GetBinaryDataLogic (const USART_TypeDef * USARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_BINARY_LOGIC_POSITIVE
  – LL_USART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_USART_GetBinaryDataLogic

LL_USART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_USART_SetTransferBitOrder (USART_TypeDef * USARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_SetTransferBitOrder

LL_USART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_USART_GetTransferBitOrder (const USART_TypeDef * USARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_BITORDER_LSBFIRST
  - LL_USART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_USART_GetTransferBitOrder

LL_USART_EnableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_EnableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Enable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_EnableAutoBaudRate

LL_USART_DisableAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_DisableAutoBaudRate (USART_TypeDef * USARTx)

Function description

Disable Auto Baud-Rate Detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_DisableAutoBaudRate

LL_USART_IsEnabledAutoBaud

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledAutoBaud (const USART_TypeDef * USARTx)

Function description

Indicate if Auto Baud-Rate Detection mechanism is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABREN LL_USART_IsEnabledAutoBaud

LL_USART_SetAutoBaudRateMode

Function name

__STATIC_INLINE void LL_USART_SetAutoBaudRateMode (USART_TypeDef * USARTx, uint32_t AutoBaudRateMode)

Function description

Set Auto Baud-Rate mode bits.

Parameters

- **USARTx:** USART Instance
- **AutoBaudRateMode:** This parameter can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_SetAutoBaudRateMode

LL_USART_GetAutoBaudRateMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetAutoBaudRateMode (const USART_TypeDef * USARTx)

Function description

Return Auto Baud-Rate mode.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_AUTOBAUD_DETECT_ON_STARTBIT
  - LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE
  - LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME
  - LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 ABRMODE LL_USART_GetAutoBaudRateMode

LL_USART_EnableRxTimeout

Function name

__STATIC_INLINE void LL_USART_EnableRxTimeout (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_EnableRxTimeout

LL_USART_DisableRxTimeout

Function name

__STATIC_INLINE void LL_USART_DisableRxTimeout (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_DisableRxTimeout

LL_USART_IsEnabledRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledRxTimeout (const USART_TypeDef * USARTx)

Function description

Indicate if Receiver Timeout feature is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RTOEN LL_USART_IsEnabledRxTimeout

LL_USART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_USART_ConfigNodeAddress (USART_TypeDef * USARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the USART node.

Parameters

- **USARTx:** USART Instance
- **AddressLen:** This parameter can be one of the following values:
  – LL_USART_ADDRESS_DETECT_4B
  – LL_USART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the USART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_ConfigNodeAddress
- CR2 ADDM7 LL_USART_ConfigNodeAddress

LL_USART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddress (const USART_TypeDef * USARTx)

Function description

Return 8 bit Address of the USART node as set in ADD field of CR2.

Parameters

- **USARTx:** USART Instance

Return values

- **Address:** of the USART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_USART_GetNodeAddress

LL_USART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetNodeAddressLen (const USART_TypeDef * USARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_ADDRESS_DETECT_4B
  – LL_USART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_USART_GetNodeAddressLen

LL_USART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_EnableRTSHWFlowCtrl

LL_USART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableRTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_DisableRTSHWFlowCtrl

LL_USART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_EnableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_EnableCTSHWFlowCtrl

LL_USART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_DisableCTSHWFlowCtrl (USART_TypeDef * USARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSE LL_USART_DisableCTSHWFlowCtrl

LL_USART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_USART_SetHWFlowCtrl (USART_TypeDef * USARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **USARTx:** USART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  – LL_USART_HWCONTROL_NONE
  – LL_USART_HWCONTROL_RTS
  – LL_USART_HWCONTROL_CTS
  – LL_USART_HWCONTROL_RTS_CTS

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_SetHWFlowCtrl
- CR3 CTSE LL_USART_SetHWFlowCtrl

LL_USART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_USART_GetHWFlowCtrl (const USART_TypeDef * USARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_HWCONTROL_NONE
  - LL_USART_HWCONTROL_RTS
  - LL_USART_HWCONTROL_CTS
  - LL_USART_HWCONTROL_RTS_CTS

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RTSE LL_USART_GetHWFlowCtrl
- CR3 CTSE LL_USART_GetHWFlowCtrl

LL_USART_EnableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_EnableOneBitSamp (USART_TypeDef * USARTx)

Function description

Enable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_EnableOneBitSamp

LL_USART_DisableOneBitSamp

Function name

__STATIC_INLINE void LL_USART_DisableOneBitSamp (USART_TypeDef * USARTx)

Function description

Disable One bit sampling method.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_DisableOneBitSamp

LL_USART_IsEnabledOneBitSamp

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOneBitSamp (const USART_TypeDef * USARTx)

Function description

Indicate if One bit sampling method is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 ONEBIT LL_USART_IsEnabledOneBitSamp

LL_USART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_EnableOverrunDetect (USART_TypeDef * USARTx)

Function description

Enable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_EnableOverrunDetect

LL_USART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_USART_DisableOverrunDetect (USART_TypeDef * USARTx)

Function description

Disable Overrun detection.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_DisableOverrunDetect

LL_USART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledOverrunDetect (const USART_TypeDef * USARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_USART_IsEnabledOverrunDetect

LL_USART_SetBaudRate

Function name

__STATIC_INLINE void LL_USART_SetBaudRate (USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling, uint32_t BaudRate)

Function description

Configure USART BRR register for achieving expected Baud Rate value.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set USARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock, Oversampling mode, and expected Baud Rate values
- Peripheral clock and Baud rate values provided as function parameters should be valid (Baud rate value != 0)
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_SetBaudRate

LL_USART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_USART_GetBaudRate (const USART_TypeDef * USARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t OverSampling)

Function description

Return current Baud Rate value, according to USARTDIV present in BRR register (full BRR content), and to used Peripheral Clock and Oversampling mode values.

Parameters

- **USARTx:** USART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_USART_PRESCALER_DIV1
  - LL_USART_PRESCALER_DIV2
  - LL_USART_PRESCALER_DIV4
  - LL_USART_PRESCALER_DIV6
  - LL_USART_PRESCALER_DIV8
  - LL_USART_PRESCALER_DIV10
  - LL_USART_PRESCALER_DIV12
  - LL_USART_PRESCALER_DIV16
  - LL_USART_PRESCALER_DIV32
  - LL_USART_PRESCALER_DIV64
  - LL_USART_PRESCALER_DIV128
  - LL_USART_PRESCALER_DIV256
- **OverSampling:** This parameter can be one of the following values:
  - LL_USART_OVERSAMPLING_16
  - LL_USART_OVERSAMPLING_8

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.
- In case of oversampling by 16 and 8, BRR content must be greater than or equal to 16d.

Reference Manual to LL API cross reference:

- BRR BRR LL_USART_GetBaudRate

LL_USART_SetRxTimeout

Function name

__STATIC_INLINE void LL_USART_SetRxTimeout (USART_TypeDef * USARTx, uint32_t Timeout)

Function description

Set Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance
- **Timeout:** Value between Min_Data=0x00 and Max_Data=0x00FFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_SetRxTimeout

LL_USART_GetRxTimeout

Function name

__STATIC_INLINE uint32_t LL_USART_GetRxTimeout (const USART_TypeDef * USARTx)

Function description

Get Receiver Time Out Value (expressed in nb of bits duration)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x00FFFFFF

Reference Manual to LL API cross reference:

- RTOR RTO LL_USART_GetRxTimeout

LL_USART_SetBlockLength

Function name

__STATIC_INLINE void LL_USART_SetBlockLength (USART_TypeDef * USARTx, uint32_t BlockLength)

Function description

Set Block Length value in reception.

Parameters

- **USARTx:** USART Instance
- **BlockLength:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_SetBlockLength

LL_USART_GetBlockLength

Function name

__STATIC_INLINE uint32_t LL_USART_GetBlockLength (const USART_TypeDef * USARTx)

Function description

Get Block Length value in reception.

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RTOR BLEN LL_USART_GetBlockLength

LL_USART_EnableIrda

Function name

__STATIC_INLINE void LL_USART_EnableIrda (USART_TypeDef * USARTx)

Function description

Enable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_EnableIrda

LL_USART_DisableIrda

Function name

__STATIC_INLINE void LL_USART_DisableIrda (USART_TypeDef * USARTx)

Function description

Disable IrDA mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_DisableIrda

LL_USART_IsEnabledIrda

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIrda (const USART_TypeDef * USARTx)

Function description

Indicate if IrDA mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IREN LL_USART_IsEnabledIrda

LL_USART_SetIrdaPowerMode

Function name

__STATIC_INLINE void LL_USART_SetIrdaPowerMode (USART_TypeDef * USARTx, uint32_t PowerMode)

Function description

Configure IrDA Power Mode (Normal or Low Power)

Parameters

- **USARTx:** USART Instance
- **PowerMode:** This parameter can be one of the following values:
  – LL_USART_IRDA_POWER_NORMAL
  – LL_USART_IRDA_POWER_LOW

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_SetIrdaPowerMode

LL_USART_GetIrdaPowerMode

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPowerMode (const USART_TypeDef * USARTx)

Function description

Retrieve IrDA Power Mode configuration (Normal or Low Power)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_IRDA_POWER_NORMAL
  – LL_USART_PHASE_2EDGE

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 IRLP LL_USART_GetIrdaPowerMode

LL_USART_SetIrdaPrescaler

Function name

__STATIC_INLINE void LL_USART_SetIrdaPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetIrdaPrescaler

LL_USART_GetIrdaPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetIrdaPrescaler (const USART_TypeDef * USARTx)

Function description

Return Irda prescaler value, used for dividing the USART clock source to achieve the Irda Low Power frequency (8 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Irda:** prescaler value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetIrdaPrescaler

LL_USART_EnableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_EnableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Enable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_EnableSmartcardNACK

LL_USART_DisableSmartcardNACK

Function name

__STATIC_INLINE void LL_USART_DisableSmartcardNACK (USART_TypeDef * USARTx)

Function description

Disable Smartcard NACK transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_DisableSmartcardNACK

LL_USART_IsEnabledSmartcardNACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcardNACK (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard NACK transmission is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 NACK LL_USART_IsEnabledSmartcardNACK

LL_USART_EnableSmartcard

Function name

__STATIC_INLINE void LL_USART_EnableSmartcard (USART_TypeDef * USARTx)

Function description

Enable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_EnableSmartcard

LL_USART_DisableSmartcard

Function name

__STATIC_INLINE void LL_USART_DisableSmartcard (USART_TypeDef * USARTx)

Function description

Disable Smartcard mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_DisableSmartcard

LL_USART_IsEnabledSmartcard

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSmartcard (const USART_TypeDef * USARTx)

Function description

Indicate if Smartcard mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCEN LL_USART_IsEnabledSmartcard

LL_USART_SetSmartcardAutoRetryCount

Function name

__STATIC_INLINE void LL_USART_SetSmartcardAutoRetryCount (USART_TypeDef * USARTx, uint32_t AutoRetryCount)

Function description

Set Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance
- **AutoRetryCount:** Value between Min_Data=0 and Max_Data=7

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- This bit-field specifies the number of retries in transmit and receive, in Smartcard mode. In transmission mode, it specifies the number of automatic retransmission retries, before generating a transmission error (FE bit set). In reception mode, it specifies the number or erroneous reception trials, before generating a reception error (RXNE and PE bits set)

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_SetSmartcardAutoRetryCount

LL_USART_GetSmartcardAutoRetryCount

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardAutoRetryCount (const USART_TypeDef * USARTx)

Function description

Return Smartcard Auto-Retry Count value (SCARCNT[2:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Auto-Retry Count value (Value between Min_Data=0 and Max_Data=7)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 SCARCNT LL_USART_GetSmartcardAutoRetryCount

LL_USART_SetSmartcardPrescaler

Function name

__STATIC_INLINE void LL_USART_SetSmartcardPrescaler (USART_TypeDef * USARTx, uint32_t PrescalerValue)

Function description

Set Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance
- **PrescalerValue:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_SetSmartcardPrescaler

LL_USART_GetSmartcardPrescaler

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardPrescaler (const USART_TypeDef * USARTx)

Function description

Return Smartcard prescaler value, used for dividing the USART clock source to provide the SMARTCARD Clock (5 bits value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** prescaler value (Value between Min_Data=0 and Max_Data=31)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR PSC LL_USART_GetSmartcardPrescaler

LL_USART_SetSmartcardGuardTime

Function name

__STATIC_INLINE void LL_USART_SetSmartcardGuardTime (USART_TypeDef * USARTx, uint32_t GuardTime)

Function description

Set Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance
- **GuardTime:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_SetSmartcardGuardTime

LL_USART_GetSmartcardGuardTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetSmartcardGuardTime (const USART_TypeDef * USARTx)

Function description

Return Smartcard Guard time value, expressed in nb of baud clocks periods (GT[7:0] bits : Guard time value)

Parameters

- **USARTx:** USART Instance

Return values

- **Smartcard:** Guard time value (Value between Min_Data=0x00 and Max_Data=0xFF)

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- GTPR GT LL_USART_GetSmartcardGuardTime

LL_USART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_EnableHalfDuplex (USART_TypeDef * USARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_EnableHalfDuplex

LL_USART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_USART_DisableHalfDuplex (USART_TypeDef * USARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_DisableHalfDuplex

LL_USART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledHalfDuplex (const USART_TypeDef * USARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_USART_IsEnabledHalfDuplex

LL_USART_EnableSPISlave

Function name

__STATIC_INLINE void LL_USART_EnableSPISlave (USART_TypeDef * USARTx)

Function description

Enable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_EnableSPISlave

LL_USART_DisableSPISlave

Function name

__STATIC_INLINE void LL_USART_DisableSPISlave (USART_TypeDef * USARTx)

Function description

Disable SPI Synchronous Slave mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_DisableSPISlave

LL_USART_IsEnabledSPISlave

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlave (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Synchronous Slave mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 SLVEN LL_USART_IsEnabledSPISlave

LL_USART_EnableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_EnableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Enable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave Selection depends on NSS input pin (The slave is selected when NSS is low and deselected when NSS is high).

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_EnableSPISlaveSelect

LL_USART_DisableSPISlaveSelect

Function name

__STATIC_INLINE void LL_USART_DisableSPISlaveSelect (USART_TypeDef * USARTx)

Function description

Disable SPI Slave Selection using NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.
- SPI Slave will be always selected and NSS input pin will be ignored.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_DisableSPISlaveSelect

LL_USART_IsEnabledSPISlaveSelect

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledSPISlaveSelect (const USART_TypeDef * USARTx)

Function description

Indicate if SPI Slave Selection depends on NSS input pin.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 DIS_NSS LL_USART_IsEnabledSPISlaveSelect

LL_USART_SetLINBrkDetectionLen

Function name

__STATIC_INLINE void LL_USART_SetLINBrkDetectionLen (USART_TypeDef * USARTx, uint32_t LINBDLength)

Function description

Set LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance
- **LINBDLength:** This parameter can be one of the following values:
  – LL_USART_LINBREAK_DETECT_10B
  – LL_USART_LINBREAK_DETECT_11B

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_SetLINBrkDetectionLen

LL_USART_GetLINBrkDetectionLen

Function name

__STATIC_INLINE uint32_t LL_USART_GetLINBrkDetectionLen (const USART_TypeDef * USARTx)

Function description

Return LIN Break Detection Length.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_LINBREAK_DETECT_10B
  - LL_USART_LINBREAK_DETECT_11B

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDL LL_USART_GetLINBrkDetectionLen

LL_USART_EnableLIN

Function name

__STATIC_INLINE void LL_USART_EnableLIN (USART_TypeDef * USARTx)

Function description

Enable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_EnableLIN

LL_USART_DisableLIN

Function name

__STATIC_INLINE void LL_USART_DisableLIN (USART_TypeDef * USARTx)

Function description

Disable LIN mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_DisableLIN

LL_USART_IsEnabledLIN

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledLIN (const USART_TypeDef * USARTx)

Function description

Indicate if LIN mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_IsEnabledLIN

LL_USART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEDeassertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_SetDEDeassertionTime

LL_USART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEDeassertionTime (const USART_TypeDef * USARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEDT LL_USART_GetDEDeassertionTime

LL_USART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_USART_SetDEAssertionTime (USART_TypeDef * USARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **USARTx:** USART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_SetDEAssertionTime

LL_USART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_USART_GetDEAssertionTime (const USART_TypeDef * USARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **USARTx:** USART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Value between Min_Data=0 and Max_Data=31

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 DEAT LL_USART_GetDEAssertionTime

LL_USART_EnableDEMode

Function name

__STATIC_INLINE void LL_USART_EnableDEMode (USART_TypeDef * USARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_EnableDEMode

LL_USART_DisableDEMode

Function name

__STATIC_INLINE void LL_USART_DisableDEMode (USART_TypeDef * USARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_DisableDEMode

LL_USART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDEMode (const USART_TypeDef * USARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEM LL_USART_IsEnabledDEMode

LL_USART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_USART_SetDESignalPolarity (USART_TypeDef * USARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance
- **Polarity:** This parameter can be one of the following values:
  – LL_USART_DE_POLARITY_HIGH
  – LL_USART_DE_POLARITY_LOW

Return values

- **None:**

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_SetDESignalPolarity

LL_USART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetDESignalPolarity (const USART_TypeDef * USARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_DE_POLARITY_HIGH
  – LL_USART_DE_POLARITY_LOW

Notes

- Macro IS_UART_DRIVER_ENABLE_INSTANCE(USARTx) can be used to check whether or not Driver Enable feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 DEP LL_USART_GetDESignalPolarity

LL_USART_ConfigAsyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigAsyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Asynchronous Mode (UART)

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In UART mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Asynchronous Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigAsyncMode
- CR2 CLKEN LL_USART_ConfigAsyncMode
- CR3 SCEN LL_USART_ConfigAsyncMode
- CR3 IREN LL_USART_ConfigAsyncMode
- CR3 HDSEL LL_USART_ConfigAsyncMode

LL_USART_ConfigSyncMode

Function name

__STATIC_INLINE void LL_USART_ConfigSyncMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Synchronous Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Synchronous mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the USART in Synchronous mode.
- Macro IS_USART_INSTANCE(USARTx) can be used to check whether or not Synchronous mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() function
- Other remaining configurations items related to Synchronous Mode (as Baud Rate, Word length, Parity, Clock Polarity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSyncMode
- CR2 CLKEN LL_USART_ConfigSyncMode
- CR3 SCEN LL_USART_ConfigSyncMode
- CR3 IREN LL_USART_ConfigSyncMode
- CR3 HDSEL LL_USART_ConfigSyncMode

LL_USART_ConfigLINMode

Function name

__STATIC_INLINE void LL_USART_ConfigLINMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in LIN Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In LIN mode, the following bits must be kept cleared: STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also set the UART/USART in LIN mode.
- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear STOP in CR2 using LL_USART_SetStopBitsLength() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionSet LINEN in CR2 using LL_USART_EnableLIN() function
- Other remaining configurations items related to LIN Mode (as Baud Rate, Word length, LIN Break Detection Length, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 CLKEN LL_USART_ConfigLINMode
- CR2 STOP LL_USART_ConfigLINMode
- CR2 LINEN LL_USART_ConfigLINMode
- CR3 IREN LL_USART_ConfigLINMode
- CR3 SCEN LL_USART_ConfigLINMode
- CR3 HDSEL LL_USART_ConfigLINMode

LL_USART_ConfigHalfDuplexMode

Function name

__STATIC_INLINE void LL_USART_ConfigHalfDuplexMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Half Duplex Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Half Duplex mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register, This function also sets the UART/USART in Half Duplex mode.
- Macro IS_UART_HALFDUPLEX_INSTANCE(USARTx) can be used to check whether or not Half-Duplex mode is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionSet HDSEL in CR3 using LL_USART_EnableHalfDuplex() function
- Other remaining configurations items related to Half Duplex Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigHalfDuplexMode
- CR2 CLKEN LL_USART_ConfigHalfDuplexMode
- CR3 HDSEL LL_USART_ConfigHalfDuplexMode
- CR3 SCEN LL_USART_ConfigHalfDuplexMode
- CR3 IREN LL_USART_ConfigHalfDuplexMode

LL_USART_ConfigSmartcardMode

Function name

__STATIC_INLINE void LL_USART_ConfigSmartcardMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Smartcard Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In Smartcard mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also configures Stop bits to 1.5 bits and sets the USART in Smartcard mode (SCEN bit). Clock Output is also enabled (CLKEN).
- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet CLKEN in CR2 using LL_USART_EnableSCLKOutput() functionSet SCEN in CR3 using LL_USART_EnableSmartcard() function
- Other remaining configurations items related to Smartcard Mode (as Baud Rate, Word length, Parity, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigSmartcardMode
- CR2 STOP LL_USART_ConfigSmartcardMode
- CR2 CLKEN LL_USART_ConfigSmartcardMode
- CR3 HDSEL LL_USART_ConfigSmartcardMode
- CR3 SCEN LL_USART_ConfigSmartcardMode

LL_USART_ConfigIrdaMode

Function name

__STATIC_INLINE void LL_USART_ConfigIrdaMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Irda Mode.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In IRDA mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,STOP and CLKEN bits in the USART_CR2 register,SCEN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register. This function also sets the UART/USART in IRDA mode (IREN bit).
- Macro IS_IRDA_INSTANCE(USARTx) can be used to check whether or not IrDA feature is supported by the USARTx instance.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() functionConfigure STOP in CR2 using LL_USART_SetStopBitsLength() functionSet IREN in CR3 using LL_USART_EnableIrda() function
- Other remaining configurations items related to Irda Mode (as Baud Rate, Word length, Power mode, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigIrdaMode
- CR2 CLKEN LL_USART_ConfigIrdaMode
- CR2 STOP LL_USART_ConfigIrdaMode
- CR3 SCEN LL_USART_ConfigIrdaMode
- CR3 HDSEL LL_USART_ConfigIrdaMode
- CR3 IREN LL_USART_ConfigIrdaMode

LL_USART_ConfigMultiProcessMode

Function name

__STATIC_INLINE void LL_USART_ConfigMultiProcessMode (USART_TypeDef * USARTx)

Function description

Perform basic configuration of USART for enabling use in Multi processor Mode (several USARTs connected in a network, one of the USARTs can be the master, its TX output connected to the RX inputs of the other slaves USARTs).

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- In MultiProcessor mode, the following bits must be kept cleared: LINEN bit in the USART_CR2 register,CLKEN bit in the USART_CR2 register,SCEN bit in the USART_CR3 register,IREN bit in the USART_CR3 register,HDSEL bit in the USART_CR3 register.
- Call of this function is equivalent to following function call sequence : Clear LINEN in CR2 using LL_USART_DisableLIN() functionClear CLKEN in CR2 using LL_USART_DisableSCLKOutput() functionClear SCEN in CR3 using LL_USART_DisableSmartcard() functionClear IREN in CR3 using LL_USART_DisableIrda() functionClear HDSEL in CR3 using LL_USART_DisableHalfDuplex() function
- Other remaining configurations items related to Multi processor Mode (as Baud Rate, Wake Up Method, Node address, ...) should be set using dedicated functions

Reference Manual to LL API cross reference:

- CR2 LINEN LL_USART_ConfigMultiProcessMode
- CR2 CLKEN LL_USART_ConfigMultiProcessMode
- CR3 SCEN LL_USART_ConfigMultiProcessMode
- CR3 HDSEL LL_USART_ConfigMultiProcessMode
- CR3 IREN LL_USART_ConfigMultiProcessMode

LL_USART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- ISR PE LL_USART_IsActiveFlag_PE

LL_USART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_FE (const USART_TypeDef * USARTx)

Function description

Check if the USART Framing Error Flag is set or not.

Parameters
- **USARTx:** USART Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- ISR FE LL_USART_IsActiveFlag_FE

LL_USART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_NE (const USART_TypeDef * USARTx)

Function description

Check if the USART Noise error detected Flag is set or not.

Parameters
- **USARTx:** USART Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- ISR NE LL_USART_IsActiveFlag_NE

LL_USART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ORE (const USART_TypeDef * USARTx)

Function description

Check if the USART OverRun Error Flag is set or not.

Parameters
- **USARTx:** USART Instance

Return values
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:
- ISR ORE LL_USART_IsActiveFlag_ORE

LL_USART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE line detected Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR IDLE LL_USART_IsActiveFlag_IDLE

LL_USART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART Read Data Register or USART RX FIFO Not Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_USART_IsActiveFlag_RXNE_RXFNE

LL_USART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TC LL_USART_IsActiveFlag_TC

LL_USART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Data Register Empty or USART TX FIFO Not Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_USART_IsActiveFlag_TXE_TXFNF

LL_USART_IsActiveFlag_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR LBDF LL_USART_IsActiveFlag_LBD

LL_USART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_nCTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS interrupt Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTSIF LL_USART_IsActiveFlag_nCTS

LL_USART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR CTS LL_USART_IsActiveFlag_CTS

LL_USART_IsActiveFlag_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Time Out Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RTOF LL_USART_IsActiveFlag_RTO

LL_USART_IsActiveFlag_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR EOBF LL_USART_IsActiveFlag_EOB

LL_USART_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_UDR (const USART_TypeDef * USARTx)

Function description

Check if the SPI Slave Underrun error flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR UDR LL_USART_IsActiveFlag_UDR

LL_USART_IsActiveFlag_ABRE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABRE (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Error Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRE LL_USART_IsActiveFlag_ABRE

LL_USART_IsActiveFlag_ABR

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_ABR (const USART_TypeDef * USARTx)

Function description

Check if the USART Auto-Baud Rate Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR ABRF LL_USART_IsActiveFlag_ABR

LL_USART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_BUSY (const USART_TypeDef * USARTx)

Function description

Check if the USART Busy Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR BUSY LL_USART_IsActiveFlag_BUSY

LL_USART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CMF LL_USART_IsActiveFlag_CM

LL_USART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_SBK (const USART_TypeDef * USARTx)

Function description

Check if the USART Send Break Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR SBKF LL_USART_IsActiveFlag_SBK

LL_USART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RWU (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR RWU LL_USART_IsActiveFlag_RWU

LL_USART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TEACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEACK LL_USART_IsActiveFlag_TEACK

LL_USART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_REACK (const USART_TypeDef * USARTx)

Function description

Check if the USART Receive Enable Acknowledge Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR REACK LL_USART_IsActiveFlag_REACK

LL_USART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFE LL_USART_IsActiveFlag_TXFE

LL_USART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFF LL_USART_IsActiveFlag_RXFF

LL_USART_IsActiveFlag_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCBGT LL_USART_IsActiveFlag_TCBGT

LL_USART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_TXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR TXFT LL_USART_IsActiveFlag_TXFT

LL_USART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_RXFT (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Threshold Flag is set or not.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ISR RXFT LL_USART_IsActiveFlag_RXFT

LL_USART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_PE (USART_TypeDef * USARTx)

Function description

Clear Parity Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_USART_ClearFlag_PE

LL_USART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_FE (USART_TypeDef * USARTx)

Function description

Clear Framing Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_USART_ClearFlag_FE

LL_USART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_NE (USART_TypeDef * USARTx)

Function description

Clear Noise Error detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_USART_ClearFlag_NE

LL_USART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_ORE (USART_TypeDef * USARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_USART_ClearFlag_ORE

LL_USART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_IDLE (USART_TypeDef * USARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_USART_ClearFlag_IDLE

LL_USART_ClearFlag_TXFE

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TXFE (USART_TypeDef * USARTx)

Function description

Clear TX FIFO Empty Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR TXFECF LL_USART_ClearFlag_TXFE

LL_USART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TC (USART_TypeDef * USARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_USART_ClearFlag_TC

LL_USART_ClearFlag_TCBGT

Function name

__STATIC_INLINE void LL_USART_ClearFlag_TCBGT (USART_TypeDef * USARTx)

Function description

Clear Smartcard Transmission Complete Before Guard Time Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCBGTCF LL_USART_ClearFlag_TCBGT

LL_USART_ClearFlag_LBD

Function name

__STATIC_INLINE void LL_USART_ClearFlag_LBD (USART_TypeDef * USARTx)

Function description

Clear LIN Break Detection Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR LBDCF LL_USART_ClearFlag_LBD

LL_USART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_USART_ClearFlag_nCTS (USART_TypeDef * USARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR CTSCF LL_USART_ClearFlag_nCTS

LL_USART_ClearFlag_RTO

Function name

__STATIC_INLINE void LL_USART_ClearFlag_RTO (USART_TypeDef * USARTx)

Function description

Clear Receiver Time Out Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR RTOCF LL_USART_ClearFlag_RTO

LL_USART_ClearFlag_EOB

Function name

__STATIC_INLINE void LL_USART_ClearFlag_EOB (USART_TypeDef * USARTx)

Function description

Clear End Of Block Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR EOBCF LL_USART_ClearFlag_EOB

LL_USART_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_USART_ClearFlag_UDR (USART_TypeDef * USARTx)

Function description

Clear SPI Slave Underrun Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_SPI_SLAVE_INSTANCE(USARTx) can be used to check whether or not SPI Slave mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- ICR UDRCF LL_USART_ClearFlag_UDR

LL_USART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_USART_ClearFlag_CM (USART_TypeDef * USARTx)

Function description

Clear Character Match Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_USART_ClearFlag_CM

LL_USART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_EnableIT_IDLE (USART_TypeDef * USARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_EnableIT_IDLE

LL_USART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_EnableIT_RXNE_RXFNE

LL_USART_EnableIT_TC

Function name

__STATIC_INLINE void LL_USART_EnableIT_TC (USART_TypeDef * USARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_EnableIT_TC

LL_USART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_EnableIT_TXE_TXFNF

LL_USART_EnableIT_PE

Function name

__STATIC_INLINE void LL_USART_EnableIT_PE (USART_TypeDef * USARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_EnableIT_PE

LL_USART_EnableIT_CM

Function name

__STATIC_INLINE void LL_USART_EnableIT_CM (USART_TypeDef * USARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_EnableIT_CM

LL_USART_EnableIT_RTO

Function name

__STATIC_INLINE void LL_USART_EnableIT_RTO (USART_TypeDef * USARTx)

Function description

Enable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_EnableIT_RTO

LL_USART_EnableIT_EOB

Function name

__STATIC_INLINE void LL_USART_EnableIT_EOB (USART_TypeDef * USARTx)

Function description

Enable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_EnableIT_EOB

LL_USART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFE (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_EnableIT_TXFE

LL_USART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFF (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_EnableIT_RXFF

LL_USART_EnableIT_LBD

Function name

__STATIC_INLINE void LL_USART_EnableIT_LBD (USART_TypeDef * USARTx)

Function description

Enable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_EnableIT_LBD

LL_USART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_EnableIT_ERROR (USART_TypeDef * USARTx)

Function description

Enable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_EnableIT_ERROR

LL_USART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_USART_EnableIT_CTS (USART_TypeDef * USARTx)

Function description

Enable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_EnableIT_CTS

LL_USART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TXFT (USART_TypeDef * USARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_EnableIT_TXFT

LL_USART_EnableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_EnableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Enable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_EnableIT_TCBGT

LL_USART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_EnableIT_RXFT (USART_TypeDef * USARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_EnableIT_RXFT

LL_USART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_USART_DisableIT_IDLE (USART_TypeDef * USARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_DisableIT_IDLE

LL_USART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXNE_RXFNE (USART_TypeDef * USARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_DisableIT_RXNE_RXFNE

LL_USART_DisableIT_TC

Function name

__STATIC_INLINE void LL_USART_DisableIT_TC (USART_TypeDef * USARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_DisableIT_TC

LL_USART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXE_TXFNF (USART_TypeDef * USARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_DisableIT_TXE_TXFNF

LL_USART_DisableIT_PE

Function name

__STATIC_INLINE void LL_USART_DisableIT_PE (USART_TypeDef * USARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_DisableIT_PE

LL_USART_DisableIT_CM

Function name

__STATIC_INLINE void LL_USART_DisableIT_CM (USART_TypeDef * USARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_DisableIT_CM

LL_USART_DisableIT_RTO

Function name

__STATIC_INLINE void LL_USART_DisableIT_RTO (USART_TypeDef * USARTx)

Function description

Disable Receiver Timeout Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_DisableIT_RTO

LL_USART_DisableIT_EOB

Function name

__STATIC_INLINE void LL_USART_DisableIT_EOB (USART_TypeDef * USARTx)

Function description

Disable End Of Block Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_DisableIT_EOB

LL_USART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFE (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_DisableIT_TXFE

LL_USART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFF (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_DisableIT_RXFF

LL_USART_DisableIT_LBD

Function name

__STATIC_INLINE void LL_USART_DisableIT_LBD (USART_TypeDef * USARTx)

Function description

Disable LIN Break Detection Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_DisableIT_LBD

LL_USART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_USART_DisableIT_ERROR (USART_TypeDef * USARTx)

Function description

Disable Error Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the USARTx_ISR register). 0: Interrupt is inhibited 1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the USARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_DisableIT_ERROR

LL_USART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_USART_DisableIT_CTS (USART_TypeDef * USARTx)

Function description

Disable CTS Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_DisableIT_CTS

LL_USART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TXFT (USART_TypeDef * USARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_DisableIT_TXFT

LL_USART_DisableIT_TCBGT

Function name

__STATIC_INLINE void LL_USART_DisableIT_TCBGT (USART_TypeDef * USARTx)

Function description

Disable Smartcard Transmission Complete Before Guard Time Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_DisableIT_TCBGT

LL_USART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_USART_DisableIT_RXFT (USART_TypeDef * USARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_DisableIT_RXFT

LL_USART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_IDLE (const USART_TypeDef * USARTx)

Function description

Check if the USART IDLE Interrupt source is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_USART_IsEnabledIT_IDLE

LL_USART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * USARTx)

Function description

Check if the USART RX Not Empty and USART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_USART_IsEnabledIT_RXNE_RXFNE

LL_USART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TC (const USART_TypeDef * USARTx)

Function description

Check if the USART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 TCIE LL_USART_IsEnabledIT_TC

LL_USART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * USARTx)

Function description

Check if the USART TX Empty and USART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_USART_IsEnabledIT_TXE_TXFNF

LL_USART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_PE (const USART_TypeDef * USARTx)

Function description

Check if the USART Parity Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 PEIE LL_USART_IsEnabledIT_PE

LL_USART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CM (const USART_TypeDef * USARTx)

Function description

Check if the USART Character Match Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 CMIE LL_USART_IsEnabledIT_CM

LL_USART_IsEnabledIT_RTO

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RTO (const USART_TypeDef * USARTx)

Function description

Check if the USART Receiver Timeout Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 RTOIE LL_USART_IsEnabledIT_RTO

LL_USART_IsEnabledIT_EOB

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_EOB (const USART_TypeDef * USARTx)

Function description

Check if the USART End Of Block Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 EOBIE LL_USART_IsEnabledIT_EOB

LL_USART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFE (const USART_TypeDef * USARTx)

Function description

Check if the USART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_USART_IsEnabledIT_TXFE

LL_USART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFF (const USART_TypeDef * USARTx)

Function description

Check if the USART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_USART_IsEnabledIT_RXFF

LL_USART_IsEnabledIT_LBD

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_LBD (const USART_TypeDef * USARTx)

Function description

Check if the USART LIN Break Detection Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_LIN_INSTANCE(USARTx) can be used to check whether or not LIN feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR2 LBDIE LL_USART_IsEnabledIT_LBD

LL_USART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_ERROR (const USART_TypeDef * USARTx)

Function description

Check if the USART Error Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIE LL_USART_IsEnabledIT_ERROR

LL_USART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_CTS (const USART_TypeDef * USARTx)

Function description

Check if the USART CTS Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_HWFLOW_INSTANCE(USARTx) can be used to check whether or not Hardware Flow control feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_USART_IsEnabledIT_CTS

LL_USART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TXFT (const USART_TypeDef * USARTx)

Function description

Check if USART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_USART_IsEnabledIT_TXFT

LL_USART_IsEnabledIT_TCBGT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_TCBGT (const USART_TypeDef * USARTx)

Function description

Check if the Smartcard Transmission Complete Before Guard Time Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_SMARTCARD_INSTANCE(USARTx) can be used to check whether or not Smartcard feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 TCBGTIE LL_USART_IsEnabledIT_TCBGT

LL_USART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_RXFT (const USART_TypeDef * USARTx)

Function description

Check if USART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_USART_IsEnabledIT_RXFT

LL_USART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_EnableDMAReq_RX

LL_USART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_RX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_DisableDMAReq_RX

LL_USART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_RX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAR LL_USART_IsEnabledDMAReq_RX

LL_USART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_EnableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_EnableDMAReq_TX

LL_USART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_USART_DisableDMAReq_TX (USART_TypeDef * USARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_DisableDMAReq_TX

LL_USART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMAReq_TX (const USART_TypeDef * USARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DMAT LL_USART_IsEnabledDMAReq_TX

LL_USART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_EnableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_EnableDMADeactOnRxErr

LL_USART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_USART_DisableDMADeactOnRxErr (USART_TypeDef * USARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_DisableDMADeactOnRxErr

LL_USART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * USARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 DDRE LL_USART_IsEnabledDMADeactOnRxErr

LL_USART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_USART_DMA_GetRegAddr (const USART_TypeDef * USARTx, uint32_t Direction)

Function description

Get the data register address used for DMA transfer.

Parameters

- **USARTx:** USART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_USART_DMA_REG_DATA_TRANSMIT
  - LL_USART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_DMA_GetRegAddr
-
- TDR TDR LL_USART_DMA_GetRegAddr

LL_USART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_USART_ReceiveData8 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData8

LL_USART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_USART_ReceiveData9 (const USART_TypeDef * USARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_USART_ReceiveData9

LL_USART_TransmitData8

Function name

__STATIC_INLINE void LL_USART_TransmitData8 (USART_TypeDef * USARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData8

LL_USART_TransmitData9

Function name

__STATIC_INLINE void LL_USART_TransmitData9 (USART_TypeDef * USARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **USARTx:** USART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_USART_TransmitData9

LL_USART_RequestAutoBaudRate

Function name

__STATIC_INLINE void LL_USART_RequestAutoBaudRate (USART_TypeDef * USARTx)

Function description

Request an Automatic Baud Rate measurement on next received data frame.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_USART_AUTOBAUDRATE_DETECTION_INSTANCE(USARTx) can be used to check whether or not Auto Baud Rate detection feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR ABRRQ LL_USART_RequestAutoBaudRate

LL_USART_RequestBreakSending

Function name

__STATIC_INLINE void LL_USART_RequestBreakSending (USART_TypeDef * USARTx)

Function description

Request Break sending.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_USART_RequestBreakSending

LL_USART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_USART_RequestEnterMuteMode (USART_TypeDef * USARTx)

Function description

Put USART in mute mode and set the RWU flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_USART_RequestEnterMuteMode

LL_USART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestRxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.
- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_USART_RequestRxDataFlush

LL_USART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_USART_RequestTxDataFlush (USART_TypeDef * USARTx)

Function description

Request a Transmit data and FIFO flush.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Notes

- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_USART_RequestTxDataFlush

LL_USART_Enable_SelectedTrigger

Function name

__STATIC_INLINE void LL_USART_Enable_SelectedTrigger (USART_TypeDef * USARTx)

Function description

Enable Selected Trigger.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_USART_Enable_SelectedTrigger

LL_USART_Disable_SelectedTrigger

Function name

__STATIC_INLINE void LL_USART_Disable_SelectedTrigger (USART_TypeDef * USARTx)

Function description

Disable Selected Trigger.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_USART_Disable_SelectedTrigger

LL_USART_IsEnabled_SelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled_SelectedTrigger (const USART_TypeDef * USARTx)

Function description

Indicate if Selected Trigger is disabled or enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AUTOCR TRIGEN LL_USART_IsEnabled_SelectedTrigger

LL_USART_Enable_AutonomousSendIdleFrame

Function name

__STATIC_INLINE void LL_USART_Enable_AutonomousSendIdleFrame (USART_TypeDef * USARTx)

Function description

Enable Autonomous Send Idle Frame feature.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR IDLEDIS LL_USART_Enable_AutonomousSendIdleFrame

LL_USART_Disable_AutonomousSendIdleFrame

Function name

__STATIC_INLINE void LL_USART_Disable_AutonomousSendIdleFrame (USART_TypeDef * USARTx)

Function description

Disable Autonomous Send Idle Frame feature.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR IDLEDIS LL_USART_Disable_AutonomousSendIdleFrame

LL_USART_IsEnabled_AutonomousSendIdleFrame

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabled_AutonomousSendIdleFrame (const USART_TypeDef * USARTx)

Function description

Indicate if Autonomous send Idle Frame feature is disabled or enabled.

Parameters

- **USARTx:** USART Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- AUTOCR IDLEDIS LL_USART_IsEnabled_AutonomousSendIdleFrame

LL_USART_SetNbTxData

Function name

__STATIC_INLINE void LL_USART_SetNbTxData (USART_TypeDef * USARTx, uint32_t Nbdata)

Function description

Configure the Number of transferred data in bytes.

Parameters

- **USARTx:** USART Instance
- **Nbdata:** This parameter can be a value between 0 and 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TDN LL_USART_SetNbTxData

LL_USART_GetNbTxData

Function name

__STATIC_INLINE uint32_t LL_USART_GetNbTxData (const USART_TypeDef * USARTx)

Function description

Retrieve the Number of transferred data in bytes.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be a value between 0 and 0xFFFF

Reference Manual to LL API cross reference:

- AUTOCR TDN LL_USART_GetNbTxData

LL_USART_SetTriggerPolarity

Function name

__STATIC_INLINE void LL_USART_SetTriggerPolarity (USART_TypeDef * USARTx, uint32_t Polarity)

Function description

Set the trigger polarity.

Parameters

- **USARTx:** USART Instance
- **Polarity:** This parameter can be one of the following values:
  – LL_USART_TRIG_POLARITY_RISING
  – LL_USART_TRIG_POLARITY_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_USART_SetTriggerPolarity

LL_USART_GetTriggerPolarity

Function name

__STATIC_INLINE uint32_t LL_USART_GetTriggerPolarity (const USART_TypeDef * USARTx)

Function description

Get the trigger polarity.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_USART_TRIG_POLARITY_RISING
  – LL_USART_TRIG_POLARITY_FALLING

Reference Manual to LL API cross reference:

- AUTOCR TRIGPOL LL_USART_GetTriggerPolarity

LL_USART_SetSelectedTrigger

Function name

__STATIC_INLINE void LL_USART_SetSelectedTrigger (USART_TypeDef * USARTx, uint32_t Trigger)

Function description

Set the selected trigger.

Parameters

- **USARTx:** USART Instance
- **Trigger:** This parameter can be one of the following values:
  - LL_USART_GPDMA1_CH0_TCF_TRG
  - LL_USART_GPDMA1_CH1_TCF_TRG
  - LL_USART_GPDMA1_CH2_TCF_TRG
  - LL_USART_GPDMA1_CH3_TCF_TRG
  - LL_USART_EXTI_LINE6_TRG
  - LL_USART_EXTI_LINE9_TRG
  - LL_USART_LPTIM1_OUT_TRG
  - LL_USART_LPTIM2_OUT_TRG
  - LL_USART_COMP1_OUT_TRG
  - LL_USART_COMP2_OUT_TRG
  - LL_USART_RTC_ALRA_TRG
  - LL_USART_RTC_WUT_TRG

Return values

- **None:**

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_USART_SetSelectedTrigger

LL_USART_GetSelectedTrigger

Function name

__STATIC_INLINE uint32_t LL_USART_GetSelectedTrigger (const USART_TypeDef * USARTx)

Function description

Get the selected trigger.

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_GPDMA1_CH0_TCF_TRG
  - LL_USART_GPDMA1_CH1_TCF_TRG
  - LL_USART_GPDMA1_CH2_TCF_TRG
  - LL_USART_GPDMA1_CH3_TCF_TRG
  - LL_USART_EXTI_LINE6_TRG
  - LL_USART_EXTI_LINE9_TRG
  - LL_USART_LPTIM1_OUT_TRG
  - LL_USART_LPTIM2_OUT_TRG
  - LL_USART_COMP1_OUT_TRG
  - LL_USART_COMP2_OUT_TRG
  - LL_USART_RTC_ALRA_TRG
  - LL_USART_RTC_WUT_TRG

Reference Manual to LL API cross reference:

- AUTOCR TRIGSEL LL_USART_GetSelectedTrigger

LL_USART_DeInit

Function name

ErrorStatus LL_USART_DeInit (const USART_TypeDef * USARTx)

Function description

De-initialize USART registers (Registers restored to their default values).

Parameters

- **USARTx:** USART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are de-initialized
  - ERROR: USART registers are not de-initialized

LL_USART_Init

Function name

ErrorStatus LL_USART_Init (USART_TypeDef * USARTx, const LL_USART_InitTypeDef * USART_InitStruct)

Function description

Initialize USART registers according to the specified parameters in USART_InitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure that contains the configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: USART registers are initialized according to USART_InitStruct content
  - ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in USART_InitStruct BaudRate field, should be valid (different from 0).

LL_USART_StructInit

Function name

void LL_USART_StructInit (LL_USART_InitTypeDef * USART_InitStruct)

Function description

Set each LL_USART_InitTypeDef field to default value.

Parameters

- **USART_InitStruct:** pointer to a LL_USART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_USART_ClockInit

Function name

ErrorStatus LL_USART_ClockInit (USART_TypeDef * USARTx, const LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Initialize USART Clock related settings according to the specified parameters in the USART_ClockInitStruct.

Parameters

- **USARTx:** USART Instance
- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure that contains the Clock configuration information for the specified USART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: USART registers related to Clock settings are initialized according to USART_ClockInitStruct content
  – ERROR: Problem occurred during USART Registers initialization

Notes

- As some bits in USART configuration registers can only be written when the USART is disabled (USART_CR1_UE bit =0), USART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_USART_ClockStructInit

Function name

void LL_USART_ClockStructInit (LL_USART_ClockInitTypeDef * USART_ClockInitStruct)

Function description

Set each field of a LL_USART_ClockInitTypeDef type structure to default value.

Parameters

- **USART_ClockInitStruct:** pointer to a LL_USART_ClockInitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

121.3 USART Firmware driver defines

The following section lists the various define and macros of the module.

121.3.1 USART

USART

Address Length Detection

LL_USART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_USART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Autobaud Detection

LL_USART_AUTOBAUD_DETECT_ON_STARTBIT

Measurement of the start bit is used to detect the baud rate

LL_USART_AUTOBAUD_DETECT_ON_FALLINGEDGE

Falling edge to falling edge measurement. Received frame must start with a single bit = 1 -> Frame = Start10xxxxxx

LL_USART_AUTOBAUD_DETECT_ON_7F_FRAME

0x7F frame detection

LL_USART_AUTOBAUD_DETECT_ON_55_FRAME

0x55 frame detection

Autonomous Trigger Polarity

LL_USART_TRIG_POLARITY_RISING

USART triggered on rising edge

LL_USART_TRIG_POLARITY_FALLING

USART triggered on falling edge

Autonomous Trigger Selection

LL_USART_GPDMA1_CH0_TCF_TRG

USART GPDMA1 channel0 Internal Trigger

LL_USART_GPDMA1_CH1_TCF_TRG

USART GPDMA1 channel1 Internal Trigger

LL_USART_GPDMA1_CH2_TCF_TRG

USART GPDMA1 channel2 Internal Trigger

LL_USART_GPDMA1_CH3_TCF_TRG

USART GPDMA1 channel3 Internal Trigger

LL_USART_EXTI_LINE6_TRG

USART EXTI line 6 Internal Trigger

LL_USART_EXTI_LINE9_TRG

USART EXTI line 9 Internal Trigger

LL_USART_LPTIM1_OUT_TRG

USART LPTIM1 out Internal Trigger

LL_USART_LPTIM2_OUT_TRG

USART LPTIM2 out Internal Trigger

LL_USART_COMP1_OUT_TRG

USART COMP1 out Internal Trigger

LL_USART_COMP2_OUT_TRG

USART COMP2 out Internal Trigger

LL_USART_RTC_ALRA_TRG

USART RTC alarm Internal Trigger

LL_USART_RTC_WUT_TRG

USART RTC wakeup Internal Trigger

Binary Data Inversion

LL_USART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_USART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_USART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_USART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_USART_ICR_PECF

Parity error clear flag

LL_USART_ICR_FECF

Framing error clear flag

LL_USART_ICR_NECF

Noise error detected clear flag

LL_USART_ICR_ORECF

Overrun error clear flag

LL_USART_ICR_IDLECF

Idle line detected clear flag

LL_USART_ICR_TXFECF

TX FIFO Empty clear flag

LL_USART_ICR_TCCF

Transmission complete clear flag

LL_USART_ICR_TCBGTCF

Transmission completed before guard time clear flag

LL_USART_ICR_LBDCF

LIN break detection clear flag

LL_USART_ICR_CTSCF

CTS clear flag

LL_USART_ICR_RTOCF

Receiver timeout clear flag

LL_USART_ICR_EOBCF

End of block clear flag

LL_USART_ICR_UDRCF

SPI Slave Underrun clear flag

LL_USART_ICR_CMCF

Character match clear flag

Clock Signal

LL_USART_CLOCK_DISABLE

Clock signal not provided

LL_USART_CLOCK_ENABLE

Clock signal provided

Datawidth

LL_USART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_USART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_USART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_USART_DE_POLARITY_HIGH

DE signal is active high

LL_USART_DE_POLARITY_LOW

DE signal is active low

Communication Direction

LL_USART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_USART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_USART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_USART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_USART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_USART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_USART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_USART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_USART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_USART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_USART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_USART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_USART_ISR_PE

Parity error flag

LL_USART_ISR_FE

Framing error flag

LL_USART_ISR_NE

Noise detected flag

LL_USART_ISR_ORE

Overrun error flag

LL_USART_ISR_IDLE

Idle line detected flag

LL_USART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_USART_ISR_TC

Transmission complete flag

LL_USART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_USART_ISR_LBDF

LIN break detection flag

LL_USART_ISR_CTSIF

CTS interrupt flag

LL_USART_ISR_CTS

CTS flag

LL_USART_ISR_RTOF

Receiver timeout flag

LL_USART_ISR_EOBF

End of block flag

LL_USART_ISR_UDR

SPI Slave underrun error flag

LL_USART_ISR_ABRE

Auto baud rate error flag

LL_USART_ISR_ABRF

Auto baud rate flag

LL_USART_ISR_BUSY

Busy flag

LL_USART_ISR_CMF

Character match flag

LL_USART_ISR_SBKF

Send break flag

LL_USART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_USART_ISR_TEACK

Transmit enable acknowledge flag

LL_USART_ISR_REACK

Receive enable acknowledge flag

LL_USART_ISR_TXFE

TX FIFO empty flag

LL_USART_ISR_RXFF

RX FIFO full flag

LL_USART_ISR_TCBGT

Transmission complete before guard time completion flag

LL_USART_ISR_RXFT

RX FIFO threshold flag

LL_USART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_USART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_USART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_USART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_USART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IrDA Power

LL_USART_IRDA_POWER_NORMAL

IrDA normal power mode

LL_USART_IRDA_POWER_LOW

IrDA low power mode

IT Defines

LL_USART_CR1_IDLEIE

IDLE interrupt enable

LL_USART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_USART_CR1_TCIE

Transmission complete interrupt enable

LL_USART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_USART_CR1_PEIE

Parity error

LL_USART_CR1_CMIE

Character match interrupt enable

LL_USART_CR1_RTOIE

Receiver timeout interrupt enable

LL_USART_CR1_EOBIE

End of Block interrupt enable

LL_USART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_USART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_USART_CR2_LBDIE

LIN break detection interrupt enable

LL_USART_CR3_EIE

Error interrupt enable

LL_USART_CR3_CTSIE

CTS interrupt enable

LL_USART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_USART_CR3_TCBGTIE

Transmission complete before guard time interrupt enable

LL_USART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Last Clock Pulse

LL_USART_LASTCLKPULSE_NO_OUTPUT

The clock pulse of the last data bit is not output to the SCLK pin

LL_USART_LASTCLKPULSE_OUTPUT

The clock pulse of the last data bit is output to the SCLK pin

LIN Break Detection Length

LL_USART_LINBREAK_DETECT_10B

10-bit break detection method selected

LL_USART_LINBREAK_DETECT_11B

11-bit break detection method selected

Oversampling

LL_USART_OVERSAMPLING_16

Oversampling by 16

LL_USART_OVERSAMPLING_8

Oversampling by 8

Parity Control

LL_USART_PARITY_NONE

Parity control disabled

LL_USART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_USART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Phase

LL_USART_PHASE_1EDGE

The first clock transition is the first data capture edge

LL_USART_PHASE_2EDGE

The second clock transition is the first data capture edge

Clock Polarity

LL_USART_POLARITY_LOW

Steady low value on SCLK pin outside transmission window

LL_USART_POLARITY_HIGH

Steady high value on SCLK pin outside transmission window

Clock Source Prescaler

LL_USART_PRESCALER_DIV1

Input clock not divided

LL_USART_PRESCALER_DIV2

Input clock divided by 2

LL_USART_PRESCALER_DIV4

Input clock divided by 4

LL_USART_PRESCALER_DIV6

Input clock divided by 6

LL_USART_PRESCALER_DIV8

Input clock divided by 8

LL_USART_PRESCALER_DIV10

Input clock divided by 10

LL_USART_PRESCALER_DIV12

Input clock divided by 12

LL_USART_PRESCALER_DIV16

Input clock divided by 16

LL_USART_PRESCALER_DIV32

Input clock divided by 32

LL_USART_PRESCALER_DIV64

Input clock divided by 64

LL_USART_PRESCALER_DIV128

Input clock divided by 128

LL_USART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_USART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_USART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_USART_STOPBITS_0_5

0.5 stop bit

LL_USART_STOPBITS_1

1 stop bit

LL_USART_STOPBITS_1_5

1.5 stop bits

LL_USART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_USART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_USART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_USART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_USART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_USART_WAKEUP_IDLELINE

USART wake up from Mute mode on Idle Line

LL_USART_WAKEUP_ADDRESSMARK

USART wake up from Mute mode on Address Mark

FLAG_Management

LL_USART_IsActiveFlag_RXNE

LL_USART_IsActiveFlag_TXE

IT_Management

LL_USART_EnableIT_RXNE

LL_USART_EnableIT_TXE

LL_USART_DisableIT_RXNE

LL_USART_DisableIT_TXE

LL_USART_IsEnabledIT_RXNE

LL_USART_IsEnabledIT_TXE

Exported_Macros_Helper

__LL_USART_DIV_SAMPLING8

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 8 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  – LL_USART_PRESCALER_DIV1
  – LL_USART_PRESCALER_DIV2
  – LL_USART_PRESCALER_DIV4
  – LL_USART_PRESCALER_DIV6
  – LL_USART_PRESCALER_DIV8
  – LL_USART_PRESCALER_DIV10
  – LL_USART_PRESCALER_DIV12
  – LL_USART_PRESCALER_DIV16
  – LL_USART_PRESCALER_DIV32
  – LL_USART_PRESCALER_DIV64
  – LL_USART_PRESCALER_DIV128
  – LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_8 case

__LL_USART_DIV_SAMPLING16

Description:

- Compute USARTDIV value according to Peripheral Clock and expected Baud Rate in 16 bits sampling mode (32 bits value of USARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for USART instance
- __PRESCALER__: This parameter can be one of the following values:
  – LL_USART_PRESCALER_DIV1
  – LL_USART_PRESCALER_DIV2
  – LL_USART_PRESCALER_DIV4
  – LL_USART_PRESCALER_DIV6
  – LL_USART_PRESCALER_DIV8
  – LL_USART_PRESCALER_DIV10
  – LL_USART_PRESCALER_DIV12
  – LL_USART_PRESCALER_DIV16
  – LL_USART_PRESCALER_DIV32
  – LL_USART_PRESCALER_DIV64
  – LL_USART_PRESCALER_DIV128
  – LL_USART_PRESCALER_DIV256
- __BAUDRATE__: Baud rate value to achieve

Return value:

- USARTDIV: value to be used for BRR register filling in OverSampling_16 case

Common Write and read registers Macros

LL_USART_WriteReg

Description:

- Write a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_USART_ReadReg

Description:

- Read a value in USART register.

Parameters:

- __INSTANCE__: USART Instance
- __REG__: Register to be read

Return value:

- Register: value

122 LL UTILS Generic Driver

122.1 UTILS Firmware driver registers structures

122.1.1 LL_UTILS_PLLInitTypeDef

LL_UTILS_PLLInitTypeDef is defined in the stm32u5xx_ll_utils.h

Data Fields

- ***uint32_t PLLM***
- ***uint32_t PLLN***
- ***uint32_t PLLR***

Field Documentation

- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLM***
  Division factor for PLL VCO input clock. This parameter can be a value of ***RCC_LL_EC_PLL1MBOOST_DIV***This feature can be modified afterwards using unitary function **LL_RCC_PLL1_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLN***
  Multiplication factor for PLL VCO output clock. This parameter must be a number between Min_Data = 8 and Max_Data = 86This feature can be modified afterwards using unitary function **LL_RCC_PLL1_ConfigDomain_SYS()**.
- ***uint32_t LL_UTILS_PLLInitTypeDef::PLLR***
  Division for the main system clock. This parameter must be a number between Min_Data = 1 and Max_Data = 128This feature can be modified afterwards using unitary function **LL_RCC_PLL1_ConfigDomain_SYS()**.

122.1.2 LL_UTILS_ClkInitTypeDef

LL_UTILS_ClkInitTypeDef is defined in the stm32u5xx_ll_utils.h

Data Fields

- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***
- ***uint32_t APB3CLKDivider***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::AHBCLKDivider***
  The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of ***RCC_LL_EC_SYSCLK_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAHBPrescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB1CLKDivider***
  The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB1_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB1Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB2CLKDivider***
  The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB2_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB2Prescaler()**.
- ***uint32_t LL_UTILS_ClkInitTypeDef::APB3CLKDivider***
  The APB3 clock (PCLK3) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of ***RCC_LL_EC_APB3_DIV***This feature can be modified afterwards using unitary function **LL_RCC_SetAPB3Prescaler()**.

122.2 UTILS Firmware driver API description

The following section lists the various functions of the UTILS library.

122.2.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 and PCLK2, PCLK3 is 160000000 Hz.

This section contains the following APIs:

- *LL_SetSystemCoreClock()*
- *LL_SetFlashLatency()*
- *LL_PLL_ConfigSystemClock_MSI()*
- *LL_PLL_ConfigSystemClock_HSI()*
- *LL_PLL_ConfigSystemClock_HSE()*

122.2.2 Detailed description of functions

LL_GetUID_Word0

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word0 (void)

Function description

Get Word0 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[31:0]:** X and Y coordinates on the wafer expressed in BCD format

LL_GetUID_Word1

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word1 (void)

Function description

Get Word1 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[63:32]:** Wafer number (UID[39:32]) & LOT_NUM[23:0] (UID[63:40])

LL_GetUID_Word2

Function name

__STATIC_INLINE uint32_t LL_GetUID_Word2 (void)

Function description

Get Word2 of the unique device identifier (UID based on 96 bits)

Return values

- **UID[95:64]:** Lot number (ASCII encoded) - LOT_NUM[55:24]

LL_GetFlashSize

Function name

__STATIC_INLINE uint32_t LL_GetFlashSize (void)

Function description

Get Flash memory size.

Return values

- **FLASH_SIZE[15:0]:** Flash memory size

Notes

- This bitfield indicates the size of the device Flash memory expressed in Kbytes. As an example, 0x040 corresponds to 64 Kbytes.

LL_GetPackageType

Function name

__STATIC_INLINE uint32_t LL_GetPackageType (void)

Function description

Get Package type.

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_PACKAGETYPE_LQFP64
  - LL_UTILS_PACKAGETYPE_LQFP100
  - LL_UTILS_PACKAGETYPE_UFBGA132
  - LL_UTILS_PACKAGETYPE_LQFP144
  - LL_UTILS_PACKAGETYPE_LQFP48
  - LL_UTILS_PACKAGETYPE_UFBGA169
  - LL_UTILS_PACKAGETYPE_LQFP64_SMPS
  - LL_UTILS_PACKAGETYPE_WLSCP90_SMPS
  - LL_UTILS_PACKAGETYPE_LQFP100_SMPS
  - LL_UTILS_PACKAGETYPE_UFBGA132_SMPS
  - LL_UTILS_PACKAGETYPE_LQFP144_SMPS
  - LL_UTILS_PACKAGETYPE_LQFP48_SMPS
  - LL_UTILS_PACKAGETYPE_UFBGA169_SMPS
  - LL_UTILS_PACKAGETYPE_WLCSP144
  - LL_UTILS_PACKAGETYPE_UFBGA144
  - LL_UTILS_PACKAGETYPE_WLCSP144_SMPS
  - LL_UTILS_PACKAGETYPE_UFBGA144_SMPS
  - LL_UTILS_PACKAGETYPE_WLCSP208_SMPS
  - LL_UTILS_PACKAGETYPE_TFBGA216_SMPS

LL_InitTick

Function name

__STATIC_INLINE void LL_InitTick (uint32_t HCLKFrequency, uint32_t Ticks)

Function description

This function configures the Cortex-M SysTick source of the time base.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)
- **Ticks:** Number of ticks

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the SysTick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick

Function name

void LL_Init1msTick (uint32_t HCLKFrequency)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base with HCLK as SysTick clock source.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.
- HCLK frequency can be calculated thanks to RCC helper macro or function LL_RCC_GetSystemClocksFreq

LL_Init1msTick_HCLK_Div8

Function name

void LL_Init1msTick_HCLK_Div8 (uint32_t HCLKFrequency)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base with HCLK/8 as SysTick clock source.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_Init1msTick_LSE

Function name

void LL_Init1msTick_LSE (void)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base with LSE as SysTick clock source.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service. LSESYS needs to be enabled to get LSE working as SysTick clock source.

LL_Init1msTick_LSI

Function name

void LL_Init1msTick_LSI (void)

Function description

This function configures the Cortex-M SysTick source to have 1ms time base with LSI as SysTick clock source.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_mDelay

Function name

void LL_mDelay (uint32_t Delay)

Function description

This function provides minimum delay (in milliseconds) based on SysTick counter flag.

Parameters

- **Delay:** specifies the minimum delay time length, in milliseconds.

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid using blocking delay and use rather osDelay service.
- To respect 1ms timebase, user should call LL_Init1msTick function which will configure Systick to 1ms

LL_SetSystemCoreClock

Function name

void LL_SetSystemCoreClock (uint32_t HCLKFrequency)

Function description

This function sets directly SystemCoreClock CMSIS variable.

Parameters

- **HCLKFrequency:** HCLK frequency in Hz (can be calculated thanks to RCC helper macro)

Return values

- **None:**

Notes

- Variable can be calculated also through SystemCoreClockUpdate function.

LL_PLL_ConfigSystemClock_MSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_MSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock with MSI as clock source of the PLL.

Parameters

- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL1.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: Max frequency configuration done
  – ERROR: Max frequency configuration not done

Notes

- The application needs to ensure that PLL1, PLL2 and/or PLL3 are disabled.
- Function is based on the following formula: PLL1 output frequency = (((MSI frequency / PLL1M) * PLL1N) / PLL1R)PLL1M: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLL1VCO_input = MSI frequency / PLL1M)PLL1N: ensure that the VCO output frequency is between 4 and 512 MHz (PLL1VCO_output = PLL1VCO_input * PLL1N)PLL1R: ensure that max frequency at 160 MHz is reached (PLL1VCO_output / PLL1R)

LL_PLL_ConfigSystemClock_HSI

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSI (LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock at maximum frequency with HSI as clock source of the PLL.

Parameters

- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: Max frequency configuration done
  – ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL1, PLL2 and/or PLL3 are disabled.
- Function is based on the following formula: PLL output frequency = (((HSI frequency / PLLM) * PLLN) / PLLR)PLL1M: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLL1VCO_input = MSI frequency / PLL1M)PLL1N: ensure that the VCO output frequency is between 4 and 512 MHz (PLL1VCO_output = PLL1VCO_input * PLL1N)PLL1R: ensure that max frequency at 160 MHz is reached (PLL1VCO_output / PLL1R)

LL_PLL_ConfigSystemClock_HSE

Function name

ErrorStatus LL_PLL_ConfigSystemClock_HSE (uint32_t HSEFrequency, uint32_t HSEBypass, LL_UTILS_PLLInitTypeDef * UTILS_PLLInitStruct, LL_UTILS_ClkInitTypeDef * UTILS_ClkInitStruct)

Function description

This function configures system clock with HSE as clock source of the PLL.

Parameters

- **HSEFrequency:** Value between Min_Data = 4000000 and Max_Data = 50000000
- **HSEBypass:** This parameter can be one of the following values:
  – LL_UTILS_HSEBYPASS_ON
  – LL_UTILS_HSEBYPASS_OFF
- **UTILS_PLLInitStruct:** pointer to a LL_UTILS_PLLInitTypeDef structure that contains the configuration information for the PLL.
- **UTILS_ClkInitStruct:** pointer to a LL_UTILS_ClkInitTypeDef structure that contains the configuration information for the BUS prescalers.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: Max frequency configuration done
  – ERROR: Max frequency configuration not done

Notes

- The application need to ensure that PLL, PLLSAI1 and/or PLLSAI2 are disabled.
- Function is based on the following formula: PLL output frequency = (((HSE frequency / PLLM) * PLLN) / PLLR)PLL1M: ensure that the VCO input frequency ranges from 1 to 16 MHz (PLL1VCO_input = MSI frequency / PLL1M)PLL1N: ensure that the VCO output frequency is between 4 and 512 MHz (PLL1VCO_output = PLL1VCO_input * PLL1N)PLL1R: ensure that max frequency at 160 MHz is reached (PLL1VCO_output / PLL1R)

LL_SetFlashLatency

Function name

ErrorStatus LL_SetFlashLatency (uint32_t HCLK_Frequency)

Function description

Update number of Flash wait states in line with new frequency and current voltage range.

Parameters

- **HCLK_Frequency:** HCLK frequency

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: Latency has been modified
  – ERROR: Latency cannot be modified

122.3 UTILS Firmware driver defines

The following section lists the various define and macros of the module.

122.3.1 UTILS

UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF

HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON

HSE Bypass is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_LQFP64

LQFP64 package type

LL_UTILS_PACKAGETYPE_WLCSP72_SMPS

WLCSP72 with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP100

LQFP100 package type

LL_UTILS_PACKAGETYPE_UFBGA132

UFBGA132 package type

LL_UTILS_PACKAGETYPE_LQFP144

LQFP144 package type

LL_UTILS_PACKAGETYPE_LQFP48

LQFP48 package type

LL_UTILS_PACKAGETYPE_UFBGA169

UFBGA169 package type

LL_UTILS_PACKAGETYPE_LQFP64_SMPS

LQFP64 with internal SMPS package type

LL_UTILS_PACKAGETYPE_WLSCP90_SMPS

WLSCP90 with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP100_SMPS

LQFP100 with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA132_SMPS

UFBGA132 with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP144_SMPS

LQFP144 with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP48_SMPS

LQFP48 with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA169_SMPS

UFBGA169 with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA64

UFBGA64 package type

LL_UTILS_PACKAGETYPE_UFBGA100

UFBGA100 package type

LL_UTILS_PACKAGETYPE_LQFP100_DSI_SMPS

LQFP100 DSI with internal SMPS package type

LL_UTILS_PACKAGETYPE_LQFP144_DSI_SMPS

LQFP144 DSI with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA144_SMPS

UFBGA144 with internal SMPS package type

LL_UTILS_PACKAGETYPE_WLCSP208_SMPS

WLCSP208 with internal SMPS package type

LL_UTILS_PACKAGETYPE_TFBGA216_SMPS

TFBGA216 with internal SMPS package type

LL_UTILS_PACKAGETYPE_UFBGA100_SMPS

UFBGA100 with internal SMPS package type

LL_UTILS_PACKAGETYPE_WLCSP56_SMPS

WLCSP56 with internal SMPS package type

LL_UTILS_PACKAGETYPE_WLCSP150_SMPS

WLCSP150 or WLCSP150 DSI with internal package type

123 LL WWDG Generic Driver

123.1 WWDG Firmware driver API description

The following section lists the various functions of the WWDG library.

123.1.1 Detailed description of functions

LL_WWDG_Enable

Function name

__STATIC_INLINE void LL_WWDG_Enable (WWDG_TypeDef * WWDGx)

Function description

Enable Window Watchdog.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- It is enabled by setting the WDGA bit in the WWDG_CR register, then it cannot be disabled again except by a reset. This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_Enable

LL_WWDG_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabled (const WWDG_TypeDef * WWDGx)

Function description

Checks if Window Watchdog is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR WDGA LL_WWDG_IsEnabled

LL_WWDG_SetCounter

Function name

__STATIC_INLINE void LL_WWDG_SetCounter (WWDG_TypeDef * WWDGx, uint32_t Counter)

Function description

Set the Watchdog counter value to provided value (7-bits T[6:0])

Parameters

- **WWDGx:** WWDG Instance
- **Counter:** 0..0x7F (7 bit counter value)

Return values

- **None:**

Notes

- When writing to the WWDG_CR register, always write 1 in the MSB b6 to avoid generating an immediate reset This counter is decremented every (4096 x 2expWDGTB) PCLK cycles A reset is produced when it rolls over from 0x40 to 0x3F (bit T6 becomes cleared) Setting the counter lower then 0x40 causes an immediate reset (if WWDG enabled)

Reference Manual to LL API cross reference:

- CR T LL_WWDG_SetCounter

LL_WWDG_GetCounter

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetCounter (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Counter Value (7 bits counter value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Counter value

Reference Manual to LL API cross reference:

- CR T LL_WWDG_GetCounter

LL_WWDG_SetPrescaler

Function name

__STATIC_INLINE void LL_WWDG_SetPrescaler (WWDG_TypeDef * WWDGx, uint32_t Prescaler)

Function description

Set the time base of the prescaler (WDGTB).

Parameters

- **WWDGx:** WWDG Instance
- **Prescaler:** This parameter can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Return values

- **None:**

Notes

- Prescaler is used to apply ratio on PCLK clock, so that Watchdog counter is decremented every (4096 x 2expWDGTB) PCLK cycles

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_SetPrescaler

LL_WWDG_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetPrescaler (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Prescaler Value.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_WWDG_PRESCALER_1
  - LL_WWDG_PRESCALER_2
  - LL_WWDG_PRESCALER_4
  - LL_WWDG_PRESCALER_8
  - LL_WWDG_PRESCALER_16
  - LL_WWDG_PRESCALER_32
  - LL_WWDG_PRESCALER_64
  - LL_WWDG_PRESCALER_128

Reference Manual to LL API cross reference:

- CFR WDGTB LL_WWDG_GetPrescaler

LL_WWDG_SetWindow

Function name

__STATIC_INLINE void LL_WWDG_SetWindow (WWDG_TypeDef * WWDGx, uint32_t Window)

Function description

Set the Watchdog Window value to be compared to the downcounter (7-bits W[6:0]).

Parameters

- **WWDGx:** WWDG Instance
- **Window:** 0x00..0x7F (7 bit Window value)

Return values

- **None:**

Notes

- This window value defines when write in the WWDG_CR register to program Watchdog counter is allowed. Watchdog counter value update must occur only when the counter value is lower than the Watchdog window register value. Otherwise, a MCU reset is generated if the 7-bit Watchdog counter value (in the control register) is refreshed before the downcounter has reached the watchdog window register value. Physically is possible to set the Window lower then 0x40 but it is not recommended. To generate an immediate reset, it is possible to set the Counter lower than 0x40.

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_SetWindow

LL_WWDG_GetWindow

Function name

__STATIC_INLINE uint32_t LL_WWDG_GetWindow (const WWDG_TypeDef * WWDGx)

Function description

Return current Watchdog Window Value (7 bits value)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **7:** bit Watchdog Window value

Reference Manual to LL API cross reference:

- CFR W LL_WWDG_GetWindow

LL_WWDG_IsActiveFlag_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsActiveFlag_EWKUP (const WWDG_TypeDef * WWDGx)

Function description

Indicates if the WWDG Early Wakeup Interrupt Flag is set or not.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Notes

- This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing 0. A write of 1 has no effect. This bit is also set if the interrupt is not enabled.

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_IsActiveFlag_EWKUP

LL_WWDG_ClearFlag_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_ClearFlag_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Clear WWDG Early Wakeup Interrupt Flag (EWIF)

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR EWIF LL_WWDG_ClearFlag_EWKUP

LL_WWDG_EnableIT_EWKUP

Function name

__STATIC_INLINE void LL_WWDG_EnableIT_EWKUP (WWDG_TypeDef * WWDGx)

Function description

Enable the Early Wakeup Interrupt.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **None:**

Notes

- When set, an interrupt occurs whenever the counter reaches value 0x40. This interrupt is only cleared by hardware after a reset

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_EnableIT_EWKUP

LL_WWDG_IsEnabledIT_EWKUP

Function name

__STATIC_INLINE uint32_t LL_WWDG_IsEnabledIT_EWKUP (const WWDG_TypeDef * WWDGx)

Function description

Check if Early Wakeup Interrupt is enabled.

Parameters

- **WWDGx:** WWDG Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CFR EWI LL_WWDG_IsEnabledIT_EWKUP

123.2 WWDG Firmware driver defines

The following section lists the various define and macros of the module.

123.2.1 WWDG

WWDG

IT Defines

LL_WWDG_CFR_EWI

PRESCALER

LL_WWDG_PRESCALER_1

WWDG counter clock = (PCLK1/4096)/1

LL_WWDG_PRESCALER_2

WWDG counter clock = (PCLK1/4096)/2

LL_WWDG_PRESCALER_4

WWDG counter clock = (PCLK1/4096)/4

LL_WWDG_PRESCALER_8

WWDG counter clock = (PCLK1/4096)/8

LL_WWDG_PRESCALER_16

WWDG counter clock = (PCLK1/4096)/16

LL_WWDG_PRESCALER_32

WWDG counter clock = (PCLK1/4096)/32

LL_WWDG_PRESCALER_64

WWDG counter clock = (PCLK1/4096)/64

LL_WWDG_PRESCALER_128

WWDG counter clock = (PCLK1/4096)/128

Common Write and read registers macros

LL_WWDG_WriteReg

Description:
- Write a value in WWDG register.

Parameters:
- __INSTANCE__: WWDG Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_WWDG_ReadReg

Description:
- Read a value in WWDG register.

Parameters:
- __INSTANCE__: WWDG Instance
- __REG__: Register to be read

Return value:
- Register: value

124 Correspondence between API registers and API low-layer driver functions

This appendix contains the correspondence table between the register or bit name, as mentioned inside the reference manual of the component, and the name of the LL functions to modify or read this register or bit.

124.1 ADC

Table 25. **Correspondence between ADC registers and ADC low-layer driver functions**

Register	Field	Function
AWD2CR	AWD2CH	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
AWD3CR	AWD3CH	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
CALFACT2	LINCALFACT	LL_ADC_GetCalibrationLinearFactor
		LL_ADC_SetCalibrationLinearFactor
CCR	CKMODE	LL_ADC_GetCommonClock
		LL_ADC_SetCommonClock
	DAMDF	LL_ADC_GetMultiDMATransfer
	DELAY	LL_ADC_GetMultiTwoSamplingDelay
		LL_ADC_SetMultiTwoSamplingDelay
	DUAL	LL_ADC_GetMultimode
		LL_ADC_SetMultimode
	PRESC	LL_ADC_GetCommonClock
		LL_ADC_SetCommonClock
	VBATEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalChAdd
		LL_ADC_SetCommonPathInternalChRem
	VREFEN	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalChAdd
		LL_ADC_SetCommonPathInternalChRem
	VSENSESEL	LL_ADC_GetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalCh
		LL_ADC_SetCommonPathInternalChAdd
		LL_ADC_SetCommonPathInternalChRem
CDR	RDATA_MST	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadMultiConversionData32
	RDATA_SLV	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadMultiConversionData32
CFGR	AUTDLY	LL_ADC_GetLowPowerMode
		LL_ADC_SetLowPowerMode

Register	Field	Function
CFGR	AWD1CH	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	AWD1EN	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	AWD1SGL	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	CHSELRMOD	LL_ADC_REG_SetSequencerConfigurable
	CONT	LL_ADC_REG_GetContinuousMode
		LL_ADC_REG_SetContinuousMode
	DISCEN	LL_ADC_REG_GetSequencerDiscont
		LL_ADC_REG_SetSequencerDiscont
	DISCNUM	LL_ADC_REG_GetSequencerDiscont
		LL_ADC_REG_SetSequencerDiscont
	DMNGT	LL_ADC_REG_GetDataTransferMode
		LL_ADC_REG_SetDataTransferMode
	EXTEN	LL_ADC_REG_GetTriggerEdge
		LL_ADC_REG_GetTriggerSource
		LL_ADC_REG_IsTriggerSourceSWStart
		LL_ADC_REG_SetTriggerEdge
		LL_ADC_REG_SetTriggerSource
	EXTSEL	LL_ADC_REG_GetTriggerSource
		LL_ADC_REG_SetTriggerSource
	JAUTO	LL_ADC_INJ_GetTrigAuto
		LL_ADC_INJ_SetTrigAuto
	JAWD1EN	LL_ADC_GetAnalogWDMonitChannels
		LL_ADC_SetAnalogWDMonitChannels
	JDISCEN	LL_ADC_INJ_GetSequencerDiscont
		LL_ADC_INJ_SetSequencerDiscont
	OVRMOD	LL_ADC_REG_GetOverrun
		LL_ADC_REG_SetOverrun
	RES	LL_ADC_GetResolution
		LL_ADC_SetResolution
CFGR1	ALIGN	LL_ADC_GetDataAlignment
		LL_ADC_SetDataAlignment
	DMACFG	LL_ADC_REG_GetDMATransfer
		LL_ADC_REG_SetDMATransfer
	DMAEN	LL_ADC_REG_GetDMATransfer
		LL_ADC_REG_SetDMATransfer
	SCANDIR	LL_ADC_REG_GetSequencerScanDirection
		LL_ADC_REG_SetSequencerScanDirection
CFGR2	BULB	LL_ADC_REG_GetSamplingMode

Register	Field	Function
CFGR2	BULB	LL_ADC_REG_SetSamplingMode
	GCOMP	LL_ADC_GetGainCompensation
		LL_ADC_SetGainCompensation
	JOVSE	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	LFTRIG	LL_ADC_GetTriggerFrequencyMode
		LL_ADC_SetTriggerFrequencyMode
	OVSR	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingRatio
	OVSS	LL_ADC_ConfigOverSamplingRatioShift
		LL_ADC_GetOverSamplingShift
	ROVSE	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	ROVSM	LL_ADC_GetOverSamplingScope
		LL_ADC_SetOverSamplingScope
	SMPTRIG	LL_ADC_REG_GetSamplingMode
		LL_ADC_REG_SetSamplingMode
	SWTRIG	LL_ADC_REG_StartSamplingPhase
		LL_ADC_REG_StopSamplingPhase
	TROVS	LL_ADC_GetOverSamplingDiscont
		LL_ADC_SetOverSamplingDiscont
CHSELR	CHSEL0	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL1	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL10	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL11	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL12	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels

Register	Field	Function
CHSELR	CHSEL13	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL14	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL15	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL16	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL17	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL18	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL2	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL3	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL4	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL5	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL6	LL_ADC_REG_GetSequencerChannels

Register	Field	Function
CHSELR	CHSEL6	LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL7	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL8	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
	CHSEL9	LL_ADC_REG_GetSequencerChannels
		LL_ADC_REG_SetSequencerChAdd
		LL_ADC_REG_SetSequencerChRem
		LL_ADC_REG_SetSequencerChannels
CR	ADCAL	LL_ADC_IsCalibrationOnGoing
		LL_ADC_StartCalibration
	ADCALDIF	LL_ADC_StartCalibration
	ADCALLIN	LL_ADC_StartCalibration
	ADDIS	LL_ADC_Disable
		LL_ADC_IsDisableOngoing
	ADEN	LL_ADC_Enable
		LL_ADC_IsEnabled
	ADSTART	LL_ADC_REG_IsConversionOngoing
		LL_ADC_REG_StartConversion
	ADSTP	LL_ADC_REG_IsStopConversionOngoing
		LL_ADC_REG_StopConversion
	ADVREGEN	LL_ADC_DisableInternalRegulator
		LL_ADC_EnableInternalRegulator
		LL_ADC_IsInternalRegulatorEnabled
	DEEPPWD	LL_ADC_DisableDeepPowerDown
		LL_ADC_EnableDeepPowerDown
		LL_ADC_IsDeepPowerDownEnabled
	JADSTART	LL_ADC_INJ_IsConversionOngoing
		LL_ADC_INJ_StartConversion
	JADSTP	LL_ADC_INJ_IsStopConversionOngoing
		LL_ADC_INJ_StopConversion
DIFSEL	DIFSEL	LL_ADC_GetChannelSingleDiff
		LL_ADC_SetChannelSingleDiff
DR	RDATA	LL_ADC_DMA_GetRegAddr
		LL_ADC_REG_ReadConversionData10

Register	Field	Function
DR	RDATA	LL_ADC_REG_ReadConversionData12
		LL_ADC_REG_ReadConversionData14
		LL_ADC_REG_ReadConversionData16
		LL_ADC_REG_ReadConversionData32
		LL_ADC_REG_ReadConversionData8
GCOMP	GCOMPCOEFF	LL_ADC_GetGainCompensation
		LL_ADC_SetGainCompensation
IER	ADRDYIE	LL_ADC_DisableIT_ADRDY
		LL_ADC_EnableIT_ADRDY
		LL_ADC_IsEnabledIT_ADRDY
	AWD1IE	LL_ADC_DisableIT_AWD1
		LL_ADC_EnableIT_AWD1
		LL_ADC_IsEnabledIT_AWD1
	AWD2IE	LL_ADC_DisableIT_AWD2
		LL_ADC_EnableIT_AWD2
		LL_ADC_IsEnabledIT_AWD2
	AWD3IE	LL_ADC_DisableIT_AWD3
		LL_ADC_EnableIT_AWD3
		LL_ADC_IsEnabledIT_AWD3
	EOCALIE	LL_ADC_DisableIT_EOCAL
		LL_ADC_EnableIT_EOCAL
		LL_ADC_IsEnabledIT_EOCAL
	EOCIE	LL_ADC_DisableIT_EOC
		LL_ADC_EnableIT_EOC
		LL_ADC_IsEnabledIT_EOC
	EOSIE	LL_ADC_DisableIT_EOS
		LL_ADC_EnableIT_EOS
		LL_ADC_IsEnabledIT_EOS
	EOSMPIE	LL_ADC_DisableIT_EOSMP
		LL_ADC_EnableIT_EOSMP
		LL_ADC_IsEnabledIT_EOSMP
	JEOCIE	LL_ADC_DisableIT_JEOC
		LL_ADC_EnableIT_JEOC
		LL_ADC_IsEnabledIT_JEOC
	JEOSIE	LL_ADC_DisableIT_JEOS
		LL_ADC_EnableIT_JEOS
		LL_ADC_IsEnabledIT_JEOS
	OVRIE	LL_ADC_DisableIT_OVR
		LL_ADC_EnableIT_OVR
		LL_ADC_IsEnabledIT_OVR
ISR	ADRDY	LL_ADC_ClearFlag_ADRDY

Register	Field	Function
ISR	ADRDY	LL_ADC_IsActiveFlag_ADRDY
	AWD1	LL_ADC_ClearFlag_AWD1
		LL_ADC_IsActiveFlag_AWD1
	AWD2	LL_ADC_ClearFlag_AWD2
		LL_ADC_IsActiveFlag_AWD2
	AWD3	LL_ADC_ClearFlag_AWD3
		LL_ADC_IsActiveFlag_AWD3
	EOC	LL_ADC_ClearFlag_EOC
		LL_ADC_IsActiveFlag_EOC
	EOS	LL_ADC_ClearFlag_EOS
		LL_ADC_IsActiveFlag_EOS
	EOSMP	LL_ADC_ClearFlag_EOSMP
		LL_ADC_IsActiveFlag_EOSMP
	JEOC	LL_ADC_ClearFlag_JEOC
		LL_ADC_IsActiveFlag_JEOC
	JEOS	LL_ADC_ClearFlag_JEOS
		LL_ADC_IsActiveFlag_JEOS
	OVR	LL_ADC_ClearFlag_OVR
		LL_ADC_IsActiveFlag_OVR
JDR1	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData14
		LL_ADC_INJ_ReadConversionData16
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData8
JDR2	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData14
		LL_ADC_INJ_ReadConversionData16
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData8
JDR3	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData14
		LL_ADC_INJ_ReadConversionData16
		LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData8
JDR4	JDATA	LL_ADC_INJ_ReadConversionData10
		LL_ADC_INJ_ReadConversionData12
		LL_ADC_INJ_ReadConversionData14
		LL_ADC_INJ_ReadConversionData16

Register	Field	Function
JDR4	JDATA	LL_ADC_INJ_ReadConversionData32
		LL_ADC_INJ_ReadConversionData8
JSQR	JEXTEN	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetTriggerEdge
		LL_ADC_INJ_GetTriggerSource
		LL_ADC_INJ_IsTriggerSourceSWStart
		LL_ADC_INJ_SetTriggerEdge
		LL_ADC_INJ_SetTriggerSource
	JEXTSEL	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetTriggerSource
		LL_ADC_INJ_SetTriggerSource
	JL	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerLength
		LL_ADC_INJ_SetSequencerLength
	JSQ1	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ2	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ3	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
	JSQ4	LL_ADC_INJ_ConfigQueueContext
		LL_ADC_INJ_GetSequencerRanks
		LL_ADC_INJ_SetSequencerRanks
OFR1	OFFSET1	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET1_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET1_EN	LL_ADC_SetOffset
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SSAT	LL_ADC_GetOffsetSignedSaturation
		LL_ADC_SetOffsetSignedSaturation
	USAT	LL_ADC_GetOffsetUnsignedSaturation
		LL_ADC_SetOffsetUnsignedSaturation
OFR2	OFFSET2	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET2_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset

Register	Field	Function
OFR2	OFFSET2_EN	LL_ADC_SetOffset
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SSAT	LL_ADC_GetOffsetSignedSaturation
		LL_ADC_SetOffsetSignedSaturation
	USAT	LL_ADC_GetOffsetUnsignedSaturation
		LL_ADC_SetOffsetUnsignedSaturation
OFR3	OFFSET3	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET3_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET3_EN	LL_ADC_SetOffset
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SSAT	LL_ADC_GetOffsetSignedSaturation
		LL_ADC_SetOffsetSignedSaturation
	USAT	LL_ADC_GetOffsetUnsignedSaturation
		LL_ADC_SetOffsetUnsignedSaturation
OFR4	OFFSET4	LL_ADC_GetOffsetLevel
		LL_ADC_SetOffset
	OFFSET4_CH	LL_ADC_GetOffsetChannel
		LL_ADC_SetOffset
	OFFSET4_EN	LL_ADC_SetOffset
	OFFSETPOS	LL_ADC_GetOffsetSign
		LL_ADC_SetOffsetSign
	SSAT	LL_ADC_GetOffsetSignedSaturation
		LL_ADC_SetOffsetSignedSaturation
	USAT	LL_ADC_GetOffsetUnsignedSaturation
		LL_ADC_SetOffsetUnsignedSaturation
PCSEL	PCSEL0	Section 90.2.1: LL_ADC_CHANNEL_0
	PCSEL1	Section 90.2.1: LL_ADC_CHANNEL_1
	PCSEL10	Section 90.2.1: LL_ADC_CHANNEL_10
	PCSEL11	Section 90.2.1: LL_ADC_CHANNEL_11
	PCSEL12	Section 90.2.1: LL_ADC_CHANNEL_12
	PCSEL13	Section 90.2.1: LL_ADC_CHANNEL_13
	PCSEL14	Section 90.2.1: LL_ADC_CHANNEL_14
	PCSEL15	Section 90.2.1: LL_ADC_CHANNEL_15
	PCSEL16	Section 90.2.1: LL_ADC_CHANNEL_16
	PCSEL17	Section 90.2.1: LL_ADC_CHANNEL_17
	PCSEL18	Section 90.2.1: LL_ADC_CHANNEL_18
	PCSEL19	Section 90.2.1: LL_ADC_CHANNEL_19

Register	Field	Function
PCSEL	PCSEL2	Section 90.2.1: LL_ADC_CHANNEL_2
	PCSEL3	Section 90.2.1: LL_ADC_CHANNEL_3
	PCSEL4	Section 90.2.1: LL_ADC_CHANNEL_4
	PCSEL5	Section 90.2.1: LL_ADC_CHANNEL_5
	PCSEL6	Section 90.2.1: LL_ADC_CHANNEL_6
	PCSEL7	Section 90.2.1: LL_ADC_CHANNEL_7
	PCSEL8	Section 90.2.1: LL_ADC_CHANNEL_8
	PCSEL9	Section 90.2.1: LL_ADC_CHANNEL_9
PW	AUTOFF	LL_ADC_GetLPModeAutoPowerOff
PWRR	AUTOFF	LL_ADC_SetLPModeAutoPowerOff
	DPD	LL_ADC_GetLPModeAutonomousDPD
		LL_ADC_SetLPModeAutonomousDPD
	VREFPROT	LL_ADC_GetVrefProtection
		LL_ADC_SetVrefProtection
	VREFSECSMP	LL_ADC_GetVrefProtection
		LL_ADC_SetVrefProtection
SMPR1	SMP0	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP1	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP2	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP3	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP4	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP5	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP6	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP7	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP8	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP9	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
SMPR2	SMP10	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP11	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP12	LL_ADC_GetChannelSamplingTime

Register	Field	Function
SMPR2	SMP12	LL_ADC_SetChannelSamplingTime
	SMP13	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP14	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP15	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP16	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP17	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
	SMP18	LL_ADC_GetChannelSamplingTime
		LL_ADC_SetChannelSamplingTime
SQR1	L	LL_ADC_REG_GetSequencerLength
		LL_ADC_REG_SetSequencerLength
	SQ1	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ2	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ3	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ4	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR2	SQ5	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ6	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ7	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ8	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ9	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR3	SQ10	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ11	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ12	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ13	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks

Register	Field	Function
SQR3	SQ14	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
SQR4	SQ15	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
	SQ16	LL_ADC_REG_GetSequencerRanks
		LL_ADC_REG_SetSequencerRanks
TR1	AWDFILT	LL_ADC_GetAWDFilteringConfiguration
		LL_ADC_SetAWDFilteringConfiguration
	HT1	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT1	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR2	HT2	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT2	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
TR3	HT3	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds
	LT3	LL_ADC_ConfigAnalogWDThresholds
		LL_ADC_GetAnalogWDThresholds
		LL_ADC_SetAnalogWDThresholds

124.2 BUS

Table 26. **Correspondence between BUS registers and BUS low-layer driver functions**

Register	Field	Function
AHB1ENR	BKPSRAMEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	CORDICEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	CRCEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DCACHE1EN	LL_AHB1_GRP1_EnableClock

Register	Field	Function
AHB1ENR	DCACHE2EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	DCACHEEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_IsEnabledClock
	DMA2DEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FLASHEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	FMACEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GFXMMUEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GPDMA1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GPU2DEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	GTZC1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	JPEGEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	MDF1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	RAMCFGEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	SRAM1EN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock
	TSCEN	LL_AHB1_GRP1_DisableClock
		LL_AHB1_GRP1_EnableClock
		LL_AHB1_GRP1_IsEnabledClock

Register	Field	Function
AHB1RSTR	CORDICRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	CRCRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	DCACHE2RSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	DMA2DRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	FMACRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GFXMMURSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPDMA1RSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	GPU2DRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	JPEGRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	MDF1RSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	RAMCFGRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
	TSCRSTR	LL_AHB1_GRP1_ForceReset
		LL_AHB1_GRP1_ReleaseReset
AHB1SMENR	BKPSRAMSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	CORDICSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	CRCSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	DCACHE2SMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	DCACHESMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	DMA2DSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep

Register	Field	Function
AHB1SMENR	DMA2DSMEN	LL_AHB1_GRP1_IsEnabledClockStopSleep
	FLASHSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	FMACSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	GFXMMUSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	GPDMA1SMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	GPU2DSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	GTZC1SMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	ICACHESMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	JPEGSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	MDF1SMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	RAMCFGSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	SRAM1SMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
	TSCSMEN	LL_AHB1_GRP1_DisableClockStopSleep
		LL_AHB1_GRP1_EnableClockStopSleep
		LL_AHB1_GRP1_IsEnabledClockStopSleep
AHB2ENR1	ADC12EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	AESEN	LL_AHB2_GRP1_DisableClock

Register	Field	Function
AHB2ENR1	AESEN	LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	DCMI_PSSIEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOAEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOBEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOCEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIODEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOEEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOFEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOGEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOHEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOIEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	GPIOJEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	HASHEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	OCTOSPIMEN	LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	OSPIMEN	LL_AHB2_GRP1_DisableClock

Register	Field	Function
AHB2ENR1	OTFDEC1EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	OTFDEC2EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	OTGEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	PKAEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	RNGEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SAESEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SDMMC1EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SDMMC2EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SRAM2EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	SRAM3EN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_IsEnabledClock
	USBPHYCEN	LL_AHB2_GRP1_DisableClock
		LL_AHB2_GRP1_EnableClock
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClock
AHB2ENR2	FSMCEN	LL_AHB2_GRP2_DisableClock
		LL_AHB2_GRP2_EnableClock
		LL_AHB2_GRP2_IsEnabledClock
	HSPI1EN	LL_AHB2_GRP2_DisableClock
		LL_AHB2_GRP2_EnableClock
		LL_AHB2_GRP2_IsEnabledClock
	OCTOSPI1EN	LL_AHB2_GRP2_DisableClock

Register	Field	Function
AHB2ENR2	OCTOSPI1EN	LL_AHB2_GRP2_EnableClock
		LL_AHB2_GRP2_IsEnabledClock
	OCTOSPI2EN	LL_AHB2_GRP2_DisableClock
		LL_AHB2_GRP2_EnableClock
		LL_AHB2_GRP2_IsEnabledClock
	SRAM5EN	LL_AHB2_GRP2_DisableClock
		LL_AHB2_GRP2_EnableClock
		LL_AHB2_GRP2_IsEnabledClock
	SRAM6EN	LL_AHB2_GRP2_DisableClock
		LL_AHB2_GRP2_EnableClock
AHB2RSTR1	ADC12RST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	AESRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	DCMI_PSSIRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOARST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOBRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOCRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIODRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOERST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOFRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOGRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOHRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOIRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	GPIOJRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	HASHRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	OCTOSPIMRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	OTFDEC1RST	LL_AHB2_GRP1_ForceReset

Register	Field	Function
AHB2RSTR1	OTFDEC1RST	LL_AHB2_GRP1_ReleaseReset
	OTFDEC2RST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	OTGRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	PKARST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	RNGRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	SAESRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	SDMMC1RST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	SDMMC2RST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
	USBPHYCRST	LL_AHB2_GRP1_ForceReset
		LL_AHB2_GRP1_ReleaseReset
AHB2RSTR2	FSMCRST	LL_AHB2_GRP2_ForceReset
		LL_AHB2_GRP2_ReleaseReset
	HSPI1RST	LL_AHB2_GRP2_ForceReset
		LL_AHB2_GRP2_ReleaseReset
	OCTOSPI1RST	LL_AHB2_GRP2_ForceReset
		LL_AHB2_GRP2_ReleaseReset
	OCTOSPI2RST	LL_AHB2_GRP2_ForceReset
		LL_AHB2_GRP2_ReleaseReset
AHB2SMENR1	ADC12SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	AESSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	DCMI_PSSISMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOASMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOBSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOCSMEN	LL_AHB2_GRP1_DisableClockStopSleep

Register	Field	Function
AHB2SMENR1	GPIOCSMEN	LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIODSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOESMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOFSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOGSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOHSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOISMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	GPIOJSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	HASHSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	OSPIMSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	OTFDEC1SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	OTFDEC2SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	OTGSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	PKASMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep

Register	Field	Function
AHB2SMENR1	RNGSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	SAESSMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	SDMMC1SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	SDMMC2SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	SRAM2SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	SRAM3SMEN	LL_AHB2_GRP1_DisableClockStopSleep
		LL_AHB2_GRP1_EnableClockStopSleep
		LL_AHB2_GRP1_IsEnabledClockStopSleep
	USBPHYCMEN	LL_AHB2_GRP1_IsEnabledClockStopSleep
	USBPHYCSMEN	LL_AHB2_GRP1_DisableClockStopSleep
AHB2SMENR2	FSMCSMEN	LL_AHB2_GRP2_DisableClockStopSleep
		LL_AHB2_GRP2_EnableClockStopSleep
		LL_AHB2_GRP2_IsEnabledClockStopSleep
	HSPI1SMEN	LL_AHB2_GRP2_DisableClockStopSleep
		LL_AHB2_GRP2_EnableClockStopSleep
		LL_AHB2_GRP2_IsEnabledClockStopSleep
	OCTOSPI1SMEN	LL_AHB2_GRP2_DisableClockStopSleep
		LL_AHB2_GRP2_EnableClockStopSleep
		LL_AHB2_GRP2_IsEnabledClockStopSleep
	OCTOSPI2SMEN	LL_AHB2_GRP2_DisableClockStopSleep
		LL_AHB2_GRP2_EnableClockStopSleep
		LL_AHB2_GRP2_IsEnabledClockStopSleep
	SRAM5SMEN	LL_AHB2_GRP2_DisableClockStopSleep
		LL_AHB2_GRP2_EnableClockStopSleep
		LL_AHB2_GRP2_IsEnabledClockStopSleep
	SRAM6SMEN	LL_AHB2_GRP2_DisableClockStopSleep
		LL_AHB2_GRP2_IsEnabledClockStopSleep
AHB3ENR	ADC4EN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	ADF1EN	LL_AHB3_GRP1_DisableClock

Register	Field	Function
AHB3ENR	ADF1EN	LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	DAC1EN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	GTZC2EN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	LPDMA1EN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	LPGPIO1EN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	PWREN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
	SRAM4EN	LL_AHB3_GRP1_DisableClock
		LL_AHB3_GRP1_EnableClock
		LL_AHB3_GRP1_IsEnabledClock
AHB3RSTR	ADC4RST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
	ADF1RST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
	DAC1RST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
	GTZC2RST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
	LPDMA1RST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
	LPGPIO1RST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
	PWRRST	LL_AHB3_GRP1_ForceReset
		LL_AHB3_GRP1_ReleaseReset
AHB3SMENR	ADC4SMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	ADF1SMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	DAC1SMEN	LL_AHB3_GRP1_DisableClockStopSleep

Register	Field	Function
AHB3SMENR	DAC1SMEN	LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	GTZC2SMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	LPDMA1SMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	LPGPIO1SMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	PWRSMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
	SRAM4SMEN	LL_AHB3_GRP1_DisableClockStopSleep
		LL_AHB3_GRP1_EnableClockStopSleep
		LL_AHB3_GRP1_IsEnabledClockStopSleep
APB1ENR1	CRSEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C1EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	I2C2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	SPI2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM3EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM4EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM5EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock

Register	Field	Function
APB1ENR1	TIM6EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	TIM7EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART4EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	UART5EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART2EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART3EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
	USART6EN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_IsEnabledClock
	USART6N	LL_APB1_GRP1_EnableClock
	WWDGEN	LL_APB1_GRP1_DisableClock
		LL_APB1_GRP1_EnableClock
		LL_APB1_GRP1_IsEnabledClock
APB1ENR2	FDCAN1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	I2C4EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	I2C5EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	I2C6EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	LPTIM2EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock
		LL_APB1_GRP2_IsEnabledClock
	UCPD1EN	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_EnableClock

Register	Field	Function
APB1ENR2	UCPD1EN	LL_APB1_GRP2_IsEnabledClock
APB1RSTR1	CRSRST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	I2C1RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	I2C2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	SPI2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM4RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM5RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM6RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	TIM7RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	UART4RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	UART5RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART2RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART3RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
	USART6RST	LL_APB1_GRP1_ForceReset
		LL_APB1_GRP1_ReleaseReset
APB1RSTR2	FDCAN1RST	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_ReleaseReset
	I2C4RST	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_ReleaseReset
	I2C5RST	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_ReleaseReset
	I2C6RST	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_ReleaseReset
	LPTIM2RST	LL_APB1_GRP2_DisableClock
		LL_APB1_GRP2_ReleaseReset

Register	Field	Function
APB1RSTR2	UCPD1RST	LL_APB1_GRP2_ReleaseReset
	UCPDRST	LL_APB1_GRP2_DisableClock
APB1SMENR1	CRSSMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	I2C1SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	I2C2SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	SPI2SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	TIM2SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	TIM3SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	TIM4SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	TIM5SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	TIM6SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	TIM7SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	UART4SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	UART5SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	USART2SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep

Register	Field	Function
APB1SMENR1	USART3SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
	USART6SMEN	LL_APB1_GRP1_DisableClockStopSleep
		LL_APB1_GRP1_EnableClockStopSleep
		LL_APB1_GRP1_IsEnabledClockStopSleep
APB1SMENR2	FDCAN1SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
		LL_APB1_GRP2_IsEnabledClockStopSleep
	I2C4SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
		LL_APB1_GRP2_IsEnabledClockStopSleep
	I2C5SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
		LL_APB1_GRP2_IsEnabledClockStopSleep
	I2C6SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
		LL_APB1_GRP2_IsEnabledClockStopSleep
	LPTIM2SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
		LL_APB1_GRP2_IsEnabledClockStopSleep
	UCPD1SMEN	LL_APB1_GRP2_DisableClockStopSleep
		LL_APB1_GRP2_EnableClockStopSleep
		LL_APB1_GRP2_IsEnabledClockStopSleep
APB2ENR	DSIHOSTEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	GFXTIMEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	LTDCEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SAI1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SAI2EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	SPI1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock

Register	Field	Function
APB2ENR	SPI1EN	LL_APB2_GRP1_IsEnabledClock
	TIM15EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM16EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM17EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	TIM8EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USART1EN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
	USBEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
APB2RSTR	DSIHOSTRST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	GFXTIMRST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	LTDCRST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SAI1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SAI2RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	SPI1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM15RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM16RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM17RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	TIM1RST	LL_APB2_GRP1_ForceReset

Register	Field	Function
APB2RSTR	TIM1RST	LL_APB2_GRP1_ReleaseReset
	TIM8RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	USART1RST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
	USBRST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset
APB2SMENR	DSIHOSTSMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	GFXTIMSMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	LTDCSMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	SAI1SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	SAI2SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	SPI1SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	TIM15SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	TIM16SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	TIM17SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	TIM1SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	TIM8SMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	USART1SMEN	LL_APB2_GRP1_DisableClockStopSleep

Register	Field	Function
APB2SMENR	USART1SMEN	LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
	USBSMEN	LL_APB2_GRP1_DisableClockStopSleep
		LL_APB2_GRP1_EnableClockStopSleep
		LL_APB2_GRP1_IsEnabledClockStopSleep
APB3ENR	COMPEN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	I2C3EN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	LPTIM1EN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	LPTIM3EN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	LPTIM4EN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	LPUART1EN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	OPAMPEN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	RTCAPBEN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	SPI3EN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	SYSCFGEN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
	VREFEN	LL_APB3_GRP1_DisableClock
		LL_APB3_GRP1_EnableClock
		LL_APB3_GRP1_IsEnabledClock
APB3RSTR	COMPRST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	I2C3RST	LL_APB3_GRP1_ForceReset

Register	Field	Function
APB3RSTR	I2C3RST	LL_APB3_GRP1_ReleaseReset
	LPTIM1RST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	LPTIM3RST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	LPTIM4RST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	LPUART1RST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	OPAMPRST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	RTCAPBRST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	SPI3RST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	SYSCFGRST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
	VREFRST	LL_APB3_GRP1_ForceReset
		LL_APB3_GRP1_ReleaseReset
APB3SMENR	COMPSMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	I2C3SMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	LPTIM1SMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	LPTIM3SMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	LPTIM4SMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	LPUART1SMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	OPAMPSMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	RTCAPBSMEN	LL_APB3_GRP1_DisableClockStopSleep

Register	Field	Function
APB3SMENR	RTCAPBSMEN	LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	SPI3SMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	SYSCFGSMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
	VREFSMEN	LL_APB3_GRP1_DisableClockStopSleep
		LL_APB3_GRP1_EnableClockStopSleep
		LL_APB3_GRP1_IsEnabledClockStopSleep
CFGR2	AHB1DIS	LL_AHB1_GRP1_DisableBusClock
		LL_AHB1_GRP1_EnableBusClock
	AHB2DIS1	LL_AHB2_GRP1_DisableBusClock
		LL_AHB2_GRP1_EnableBusClock
	AHB2DIS2	LL_AHB2_GRP2_DisableBusClock
		LL_AHB2_GRP2_EnableBusClock
	AHB3DIS	LL_AHB3_GRP1_DisableBusClock
		LL_AHB3_GRP1_EnableBusClock
	APB1DIS	LL_APB1_GRP1_DisableBusClock
		LL_APB1_GRP1_EnableBusClock
	APB2DIS	LL_APB2_GRP1_DisableBusClock
		LL_APB2_GRP1_EnableBusClock
	APB3DIS	LL_APB3_GRP1_DisableBusClock
		LL_APB3_GRP1_EnableBusClock
SRDAMR	ADC4AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	ADF1AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	COMPAMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	DAC1AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	I2C3AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	LPDMA1AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock

Register	Field	Function
SRDAMR	LPDMA1AMEN	LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	LPGPIO1AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	LPTIM1AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	LPTIM3AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	LPTIM4AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	LPUART1AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	OPAMPAMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	RTCAPBAMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	SPI3AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	SRAM4AMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	VREFAMEN	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock
	VREFRST	LL_SRDAMR_GRP1_DisableAutonomousClock
		LL_SRDAMR_GRP1_EnableAutonomousClock
		LL_SRDAMR_GRP1_IsEnabledAutonomousClock

124.3 COMP

Table 27. **Correspondence between COMP registers and COMP low-layer driver functions**

Register	Field	Function
CSR	BLANKSEL	LL_COMP_GetOutputBlankingSource

Register	Field	Function
CSR	BLANKSEL	LL_COMP_SetOutputBlankingSource
	EN	LL_COMP_Disable
		LL_COMP_Enable
		LL_COMP_IsEnabled
	HYST	LL_COMP_GetInputHysteresis
		LL_COMP_SetInputHysteresis
	INMSEL	LL_COMP_ConfigInputs
		LL_COMP_GetInputMinus
		LL_COMP_SetInputMinus
	INPSEL	LL_COMP_ConfigInputs
		LL_COMP_GetInputPlus
		LL_COMP_SetInputPlus
	LOCK	LL_COMP_IsLocked
		LL_COMP_Lock
	POLARITY	LL_COMP_GetOutputPolarity
		LL_COMP_SetOutputPolarity
	PWRMODE	LL_COMP_GetPowerMode
		LL_COMP_SetPowerMode
	VALUE	LL_COMP_ReadOutputLevel
	WINMODE	LL_COMP_GetCommonWindowMode
		LL_COMP_GetCommonWindowOutput
		LL_COMP_SetCommonWindowMode
	WINOUT	LL_COMP_SetCommonWindowOutput

124.4 CORDIC

Table 28. Correspondence between CORDIC registers and CORDIC low-layer driver functions

Register	Field	Function
CSR	ARGSIZE	LL_CORDIC_Config
		LL_CORDIC_GetInSize
		LL_CORDIC_SetInSize
	DMAREN	LL_CORDIC_DisableDMAReq_RD
		LL_CORDIC_EnableDMAReq_RD
		LL_CORDIC_IsEnabledDMAReq_RD
	DMAWEN	LL_CORDIC_DisableDMAReq_WR
		LL_CORDIC_EnableDMAReq_WR
		LL_CORDIC_IsEnabledDMAReq_WR
	FUNC	LL_CORDIC_Config
		LL_CORDIC_GetFunction
		LL_CORDIC_SetFunction
	IEN	LL_CORDIC_DisableIT

Register	Field	Function
CSR	IEN	LL_CORDIC_EnableIT
		LL_CORDIC_IsEnabledIT
	NARGS	LL_CORDIC_Config
		LL_CORDIC_GetNbWrite
		LL_CORDIC_SetNbWrite
	NRES	LL_CORDIC_Config
		LL_CORDIC_GetNbRead
		LL_CORDIC_SetNbRead
	PRECISION	LL_CORDIC_Config
		LL_CORDIC_GetPrecision
		LL_CORDIC_SetPrecision
	RESIZE	LL_CORDIC_Config
		LL_CORDIC_GetOutSize
		LL_CORDIC_SetOutSize
	RRDY	LL_CORDIC_IsActiveFlag_RRDY
	SCALE	LL_CORDIC_Config
		LL_CORDIC_GetScale
		LL_CORDIC_SetScale
RDATA	RES	LL_CORDIC_DMA_GetRegAddr
		LL_CORDIC_ReadData
WDATA	ARG	LL_CORDIC_DMA_GetRegAddr
		LL_CORDIC_WriteData

124.5 CORTEX

Table 29. Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	LL_MPU_Disable
		LL_MPU_Disable_NS
		LL_MPU_Enable
		LL_MPU_IsEnabled
		LL_MPU_IsEnabled_NS
MPU_MAIR0	Attribute	LL_MPU_ConfigAttributes
		LL_MPU_ConfigAttributes_NS
MPU_MAIR1	Attribute	LL_MPU_ConfigAttributes
		LL_MPU_ConfigAttributes_NS
MPU_RBAR	ADDR	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegionAddress
		LL_MPU_ConfigRegionAddress_NS
		LL_MPU_ConfigRegion_NS
		LL_MPU_SetRegionBaseAddress

Register	Field	Function
MPU_RBAR	ADDR	LL_MPU_SetRegionBaseAddress_NS
	AP	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
	SH	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
	XN	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
		LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
MPU_RLAR	ADDR	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegionAddress
		LL_MPU_ConfigRegionAddress_NS
		LL_MPU_ConfigRegion_NS
		LL_MPU_SetRegionLimitAddress
		LL_MPU_SetRegionLimitAddress_NS
	AttrIndx	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
	EN	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegion_NS
	ENABLE	LL_MPU_DisableRegion
		LL_MPU_DisableRegion_NS
		LL_MPU_EnableRegion
		LL_MPU_EnableRegion_NS
MPU_RNR	ENABLE	LL_MPU_IsEnabled_Region
		LL_MPU_IsEnabled_Region_NS
	REGION	LL_MPU_ConfigRegion
		LL_MPU_ConfigRegionAddress
		LL_MPU_ConfigRegionAddress_NS
		LL_MPU_ConfigRegion_NS
		LL_MPU_DisableRegion
		LL_MPU_DisableRegion_NS

Register	Field	Function
MPU_RNR	REGION	LL_MPU_GetRegionAccess
		LL_MPU_GetRegionAccess_NS
		LL_MPU_GetRegionBaseAddress
		LL_MPU_GetRegionBaseAddress_NS
		LL_MPU_GetRegionLimitAddress
		LL_MPU_GetRegionLimitAddress_NS
		LL_MPU_SetRegionAccess
		LL_MPU_SetRegionAccess_NS
		LL_MPU_SetRegionBaseAddress
		LL_MPU_SetRegionBaseAddress_NS
		LL_MPU_SetRegionLimitAddress
		LL_MPU_SetRegionLimitAddress_NS
SCB_CPUID	ARCHITECTURE	LL_CPUID_GetArchitecture
	IMPLEMENTER	LL_CPUID_GetImplementer
	PARTNO	LL_CPUID_GetParNo
	REVISION	LL_CPUID_GetRevision
	VARIANT	LL_CPUID_GetVariant
SCB_SCR	SEVEONPEND	LL_LPM_DisableEventOnPend
		LL_LPM_EnableEventOnPend
	SLEEPDEEP	LL_LPM_EnableDeepSleep
		LL_LPM_EnableSleep
	SLEEPONEXIT	LL_LPM_DisableSleepOnExit
		LL_LPM_EnableSleepOnExit
SCB_SHCSR	BUSFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
	MEMFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
	SECUREFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
	USGFAULTENA	LL_HANDLER_DisableFault
		LL_HANDLER_EnableFault
STK_CTRL	CLKSOURCE	LL_SYSTICK_GetClkSource
		LL_SYSTICK_SetClkSource
	COUNTFLAG	LL_SYSTICK_IsActiveCounterFlag
	TICKINT	LL_SYSTICK_DisableIT
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

124.6 CRC

Table 30. Correspondence between CRC registers and CRC low-layer driver functions

Register	Field	Function
CR	POLYSIZE	LL_CRC_GetPolynomialSize
		LL_CRC_SetPolynomialSize
	RESET	LL_CRC_ResetCRCCalculationUnit
	REV_IN	LL_CRC_GetInputDataReverseMode
		LL_CRC_SetInputDataReverseMode
	REV_OUT	LL_CRC_GetOutputDataReverseMode
		LL_CRC_SetOutputDataReverseMode
DR	DR	LL_CRC_FeedData16
		LL_CRC_FeedData32
		LL_CRC_FeedData8
		LL_CRC_ReadData16
		LL_CRC_ReadData32
		LL_CRC_ReadData7
		LL_CRC_ReadData8
IDR	IDR	LL_CRC_Read_IDR
		LL_CRC_Write_IDR
INIT	INIT	LL_CRC_GetInitialData
		LL_CRC_SetInitialData
POL	POL	LL_CRC_GetPolynomialCoef
		LL_CRC_SetPolynomialCoef

124.7 CRS

Table 31. Correspondence between CRS registers and CRS low-layer driver functions

Register	Field	Function
CFGR	FELIM	LL_CRS_ConfigSynchronization
		LL_CRS_GetFreqErrorLimit
		LL_CRS_SetFreqErrorLimit
	RELOAD	LL_CRS_ConfigSynchronization
		LL_CRS_GetReloadCounter
		LL_CRS_SetReloadCounter
	SYNCDIV	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncDivider
		LL_CRS_SetSyncDivider
	SYNCPOL	LL_CRS_ConfigSynchronization
		LL_CRS_GetSyncPolarity
		LL_CRS_SetSyncPolarity
	SYNCSRC	LL_CRS_ConfigSynchronization

Register	Field	Function
CFGR	SYNCSRC	LL_CRS_GetSyncSignalSource
		LL_CRS_SetSyncSignalSource
CR	AUTOTRIMEN	LL_CRS_DisableAutoTrimming
		LL_CRS_EnableAutoTrimming
		LL_CRS_IsEnabledAutoTrimming
	CEN	LL_CRS_DisableFreqErrorCounter
		LL_CRS_EnableFreqErrorCounter
		LL_CRS_IsEnabledFreqErrorCounter
	ERRIE	LL_CRS_DisableIT_ERR
		LL_CRS_EnableIT_ERR
		LL_CRS_IsEnabledIT_ERR
	ESYNCIE	LL_CRS_DisableIT_ESYNC
		LL_CRS_EnableIT_ESYNC
		LL_CRS_IsEnabledIT_ESYNC
	SWSYNC	LL_CRS_GenerateEvent_SWSYNC
	SYNCOKIE	LL_CRS_DisableIT_SYNCOK
		LL_CRS_EnableIT_SYNCOK
		LL_CRS_IsEnabledIT_SYNCOK
	SYNCWARNIE	LL_CRS_DisableIT_SYNCWARN
		LL_CRS_EnableIT_SYNCWARN
		LL_CRS_IsEnabledIT_SYNCWARN
	TRIM	LL_CRS_ConfigSynchronization
		LL_CRS_GetHSI48SmoothTrimming
		LL_CRS_SetHSI48SmoothTrimming
ICR	ERRC	LL_CRS_ClearFlag_ERR
	ESYNCC	LL_CRS_ClearFlag_ESYNC
	SYNCOKC	LL_CRS_ClearFlag_SYNCOK
	SYNCWARNC	LL_CRS_ClearFlag_SYNCWARN
ISR	ERRF	LL_CRS_IsActiveFlag_ERR
	ESYNCF	LL_CRS_IsActiveFlag_ESYNC
	FECAP	LL_CRS_GetFreqErrorCapture
	FEDIR	LL_CRS_GetFreqErrorDirection
	SYNCERR	LL_CRS_IsActiveFlag_SYNCERR
	SYNCMISS	LL_CRS_IsActiveFlag_SYNCMISS
	SYNCOKF	LL_CRS_IsActiveFlag_SYNCOK
	SYNCWARNF	LL_CRS_IsActiveFlag_SYNCWARN
	TRIMOVF	LL_CRS_IsActiveFlag_TRIMOVF

124.8 DAC

Table 32. Correspondence between DAC registers and DAC low-layer driver functions

Register	Field	Function
AUTOCR	AUTOMODE	LL_DAC_DisableAutonomousMode
		LL_DAC_EnableAutonomousMode
CCR	OTRIM1	LL_DAC_GetTrimmingValue
		LL_DAC_SetTrimmingValue
	OTRIM2	LL_DAC_GetTrimmingValue
		LL_DAC_SetTrimmingValue
CR	CEN1	LL_DAC_GetMode
		LL_DAC_SetMode
	CEN2	LL_DAC_GetMode
		LL_DAC_SetMode
	DMAEN1	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAEN2	LL_DAC_DisableDMAReq
		LL_DAC_EnableDMAReq
		LL_DAC_IsDMAReqEnabled
	DMAUDRIE1	LL_DAC_DisableIT_DMAUDR1
		LL_DAC_EnableIT_DMAUDR1
		LL_DAC_IsEnabledIT_DMAUDR1
	DMAUDRIE2	LL_DAC_DisableIT_DMAUDR2
		LL_DAC_EnableIT_DMAUDR2
		LL_DAC_IsEnabledIT_DMAUDR2
	EN1	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled
	EN2	LL_DAC_Disable
		LL_DAC_Enable
		LL_DAC_IsEnabled
	MAMP1	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MAMP2	LL_DAC_GetWaveNoiseLFSR
		LL_DAC_GetWaveTriangleAmplitude
		LL_DAC_SetWaveNoiseLFSR
		LL_DAC_SetWaveTriangleAmplitude
	MODE1	LL_DAC_ConfigOutput
		LL_DAC_GetOutputBuffer

Register	Field	Function
CR	MODE1	LL_DAC_GetOutputConnection
		LL_DAC_GetOutputMode
		LL_DAC_SetOutputBuffer
		LL_DAC_SetOutputConnection
		LL_DAC_SetOutputMode
	MODE2	LL_DAC_ConfigOutput
		LL_DAC_GetOutputBuffer
		LL_DAC_GetOutputConnection
		LL_DAC_GetOutputMode
		LL_DAC_SetOutputBuffer
		LL_DAC_SetOutputConnection
		LL_DAC_SetOutputMode
	TEN1	LL_DAC_DisableTrigger
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TEN2	LL_DAC_DisableTrigger
		LL_DAC_EnableTrigger
		LL_DAC_IsTriggerEnabled
	TSEL1	LL_DAC_GetTriggerSource
		LL_DAC_SetTriggerSource
	TSEL2	LL_DAC_GetTriggerSource
		LL_DAC_SetTriggerSource
	WAVE1	LL_DAC_GetWaveAutoGeneration
		LL_DAC_SetWaveAutoGeneration
	WAVE2	LL_DAC_GetWaveAutoGeneration
		LL_DAC_SetWaveAutoGeneration
DHR12L1	DACC1DHR	LL_DAC_ConvertData12LeftAligned
		LL_DAC_DMA_GetRegAddr
DHR12L2	DACC2DHR	LL_DAC_ConvertData12LeftAligned
		LL_DAC_DMA_GetRegAddr
DHR12LD	DACC1DHR	LL_DAC_ConvertDualData12LeftAligned
	DACC2DHR	LL_DAC_ConvertDualData12LeftAligned
DHR12R1	DACC1DHR	LL_DAC_ConvertData12RightAligned
		LL_DAC_DMA_GetRegAddr
DHR12R2	DACC2DHR	LL_DAC_ConvertData12RightAligned
		LL_DAC_DMA_GetRegAddr
DHR12RD	DACC1DHR	LL_DAC_ConvertDualData12RightAligned
	DACC2DHR	LL_DAC_ConvertDualData12RightAligned
DHR8R1	DACC1DHR	LL_DAC_ConvertData8RightAligned
		LL_DAC_DMA_GetRegAddr
DHR8R2	DACC2DHR	LL_DAC_ConvertData8RightAligned

Register	Field	Function
DHR8R2	DACC2DHR	LL_DAC_DMA_GetRegAddr
DHR8RD	DACC1DHR	LL_DAC_ConvertDualData8RightAligned
	DACC2DHR	LL_DAC_ConvertDualData8RightAligned
DOR1	DACC1DOR	LL_DAC_RetrieveOutputData
DOR2	DACC2DOR	LL_DAC_RetrieveOutputData
MCR	DMADOUBLE1	LL_DAC_DisableDMADoubleDataMode
		LL_DAC_EnableDMADoubleDataMode
		LL_DAC_IsDMADoubleDataModeEnabled
	DMADOUBLE2	LL_DAC_DisableDMADoubleDataMode
		LL_DAC_EnableDMADoubleDataMode
		LL_DAC_IsDMADoubleDataModeEnabled
	HFSEL	LL_DAC_GetHighFrequencyMode
		LL_DAC_SetHighFrequencyMode
	SINFORMAT1	LL_DAC_GetSignedFormat
		LL_DAC_SetSignedFormat
	SINFORMAT2	LL_DAC_GetSignedFormat
		LL_DAC_SetSignedFormat
SHHR	THOLD1	LL_DAC_GetSampleAndHoldHoldTime
		LL_DAC_SetSampleAndHoldHoldTime
	THOLD2	LL_DAC_GetSampleAndHoldHoldTime
		LL_DAC_SetSampleAndHoldHoldTime
SHRR	TREFRESH1	LL_DAC_GetSampleAndHoldRefreshTime
		LL_DAC_SetSampleAndHoldRefreshTime
	TREFRESH2	LL_DAC_GetSampleAndHoldRefreshTime
		LL_DAC_SetSampleAndHoldRefreshTime
SHSR1	TSAMPLE1	LL_DAC_GetSampleAndHoldSampleTime
		LL_DAC_SetSampleAndHoldSampleTime
SHSR2	TSAMPLE2	LL_DAC_GetSampleAndHoldSampleTime
		LL_DAC_SetSampleAndHoldSampleTime
SR	BWST1	LL_DAC_IsActiveFlag_BWST1
	BWST2	LL_DAC_IsActiveFlag_BWST2
	CAL_FLAG1	LL_DAC_IsActiveFlag_CAL1
	CAL_FLAG2	LL_DAC_IsActiveFlag_CAL2
	DAC1RDY	LL_DAC_IsActiveFlag_DAC1RDY
		LL_DAC_IsReady
	DAC2RDY	LL_DAC_IsActiveFlag_DAC2RDY
		LL_DAC_IsReady
	DMAUDR1	LL_DAC_ClearFlag_DMAUDR1
		LL_DAC_IsActiveFlag_DMAUDR1
	DMAUDR2	LL_DAC_ClearFlag_DMAUDR2
		LL_DAC_IsActiveFlag_DMAUDR2

Register	Field	Function
SR	DORSTAT1	LL_DAC_IsActiveFlag_DORSTAT1
	DORSTAT2	LL_DAC_IsActiveFlag_DORSTAT2
SWTRIGR	SWTRIG1	LL_DAC_TrigSWConversion
	SWTRIG2	LL_DAC_TrigSWConversion

124.9 DCACHE

Table 33. **Correspondence between DCACHE registers and DCACHE low-layer driver functions**

Register	Field	Function
CR	(RHITMRST/RMISSMRST/WHITMRST/WMISSMRST)	LL_DCACHE_ResetMonitors
	(RMISSMEN/RHITMEN/WMISSMEN/WHITMEN)	LL_DCACHE_DisableMonitors
		LL_DCACHE_EnableMonitors
		LL_DCACHE_IsEnabledMonitors
	CACHECMD	LL_DCACHE_GetCommand
		LL_DCACHE_SetCommand
		LL_DCACHE_StartCommand
	CACHEINV	LL_DCACHE_Invalidate
	CMDREADDRR	LL_DCACHE_GetEndAddress
		LL_DCACHE_SetEndAddress
	CMDRSADDRR	LL_DCACHE_GetStartAddress
		LL_DCACHE_SetStartAddress
	EN	LL_DCACHE_Disable
		LL_DCACHE_Enable
		LL_DCACHE_IsEnabled
	HBURST	LL_DCACHE_GetReadBurstType
		LL_DCACHE_SetReadBurstType
FCR	CBSYENDF	LL_DCACHE_ClearFlag_BSYEND
	CCMDENDF	LL_DCACHE_ClearFlag_CMDEND
	CERRF	LL_DCACHE_ClearFlag_ERR
IER	BSYENDIE	LL_DCACHE_DisableIT_BSYEND
		LL_DCACHE_EnableIT_BSYEND
		LL_DCACHE_IsEnabledIT_BSYEND
	CMDENDIE	LL_DCACHE_DisableIT_CMDEND
		LL_DCACHE_EnableIT_CMDEND
		LL_DCACHE_IsEnabledIT_CMDEND
	ERRIE	LL_DCACHE_DisableIT_ERR
		LL_DCACHE_EnableIT_ERR
		LL_DCACHE_IsEnabledIT_ERR
RHMONR	RHITMON	LL_DCACHE_Monitor_GetReadHitValue
RMMONR	RMISSMON	LL_DCACHE_Monitor_GetReadMissValue
SR	BSYENDF	LL_DCACHE_IsActiveFlag_BSYEND

Register	Field	Function
SR	BUSYCMDF	LL_DCACHE_IsActiveFlag_BUSYCMD
	BUSYF	LL_DCACHE_IsActiveFlag_BUSY
	CMDENDF	LL_DCACHE_IsActiveFlag_CMDEND
	ERRF	LL_DCACHE_IsActiveFlag_ERR
WHMONR	WHITMON	LL_DCACHE_Monitor_GetWriteHitValue
WMMONR	WMISSMON	LL_DCACHE_Monitor_GetWriteMissValue

124.10 DMA

Table 34. Correspondence between DMA registers and DMA low-layer driver functions

Register	Field	Function
CBR1	BNDT	LL_DMA_ConfigBlkCounters
		LL_DMA_GetBlkDataLength
		LL_DMA_SetBlkDataLength
	BRC	LL_DMA_ConfigBlkCounters
		LL_DMA_GetBlkRptCount
		LL_DMA_SetBlkRptCount
	BRDDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetBlkRptDestAddrUpdate
		LL_DMA_SetBlkRptDestAddrUpdate
	BRSDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetBlkRptSrcAddrUpdate
		LL_DMA_SetBlkRptSrcAddrUpdate
	DDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetDestAddrUpdate
		LL_DMA_SetDestAddrUpdate
	SDEC	LL_DMA_ConfigBlkRptAddrUpdate
		LL_DMA_GetSrcAddrUpdate
		LL_DMA_SetSrcAddrUpdate
CBR2	BRDAO	LL_DMA_ConfigBlkRptAddrUpdateValue
		LL_DMA_GetBlkRptDestAddrUpdateValue
		LL_DMA_SetBlkRptDestAddrUpdateValue
	BRSAO	LL_DMA_ConfigBlkRptAddrUpdateValue
		LL_DMA_GetBlkRptSrcAddrUpdateValue
		LL_DMA_SetBlkRptSrcAddrUpdateValue
CCR	DTEIE	LL_DMA_DisableIT_DTE
		LL_DMA_EnableIT_DTE
		LL_DMA_IsEnabledIT_DTE
	EN	LL_DMA_DisableChannel
		LL_DMA_EnableChannel
		LL_DMA_IsEnabledChannel

Register	Field	Function
CCR	HTIE	LL_DMA_DisableIT_HT
		LL_DMA_EnableIT_HT
		LL_DMA_IsEnabledIT_HT
	LAP	LL_DMA_ConfigControl
		LL_DMA_GetLinkAllocatedPort
		LL_DMA_SetLinkAllocatedPort
	LSM	LL_DMA_ConfigControl
		LL_DMA_GetLinkStepMode
		LL_DMA_SetLinkStepMode
	PRIO	LL_DMA_ConfigControl
		LL_DMA_GetChannelPriorityLevel
		LL_DMA_SetChannelPriorityLevel
	RESET	LL_DMA_ResetChannel
	SUSP	LL_DMA_IsSuspendedChannel
		LL_DMA_ResumeChannel
		LL_DMA_SuspendChannel
	SUSPIE	LL_DMA_DisableIT_SUSP
		LL_DMA_EnableIT_SUSP
		LL_DMA_IsEnabledIT_SUSP
	TCIE	LL_DMA_DisableIT_TC
		LL_DMA_EnableIT_TC
		LL_DMA_IsEnabledIT_TC
	TOIE	LL_DMA_DisableIT_TO
		LL_DMA_EnableIT_TO
		LL_DMA_IsEnabledIT_TO
	ULEIE	LL_DMA_DisableIT_ULE
		LL_DMA_EnableIT_ULE
		LL_DMA_IsEnabledIT_ULE
	USEIE	LL_DMA_DisableIT_USE
		LL_DMA_EnableIT_USE
		LL_DMA_IsEnabledIT_USE
CDAR	DA	LL_DMA_ConfigAddresses
		LL_DMA_GetDestAddress
		LL_DMA_SetDestAddress
	DAO	LL_DMA_GetDestAddrUpdateValue
CFCR	DTEF	LL_DMA_ClearFlag_DTE
	HTF	LL_DMA_ClearFlag_HT
	SUSPF	LL_DMA_ClearFlag_SUSP
	TCF	LL_DMA_ClearFlag_TC
	TOF	LL_DMA_ClearFlag_TO
	ULEF	LL_DMA_ClearFlag_ULE

Register	Field	Function
CFCR	USEF	LL_DMA_ClearFlag_USE
CLBAR	LBA	LL_DMA_GetLinkedListBaseAddr
		LL_DMA_SetLinkedListBaseAddr
CLLR	LA	LL_DMA_GetLinkedListAddrOffset
		LL_DMA_SetLinkedListAddrOffset
	UB1	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCBR1Update
		LL_DMA_EnableCBR1Update
		LL_DMA_IsEnabledCBR1Update
	UB2	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCBR2Update
		LL_DMA_EnableCBR2Update
		LL_DMA_IsEnabledCBR2Update
	UDA	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCDARUpdate
		LL_DMA_EnableCDARUpdate
		LL_DMA_IsEnabledCDARUpdate
	ULL	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCLLRUpdate
		LL_DMA_EnableCLLRUpdate
		LL_DMA_IsEnabledCLLRUpdate
	USA	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCSARUpdate
		LL_DMA_EnableCSARUpdate
		LL_DMA_IsEnabledCSARUpdate
	UT1	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCTR1Update
		LL_DMA_EnableCTR1Update
		LL_DMA_IsEnabledCTR1Update
	UT2	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCTR2Update
		LL_DMA_EnableCTR2Update
		LL_DMA_IsEnabledCTR2Update
	UT3	LL_DMA_ConfigLinkUpdate
		LL_DMA_DisableCTR3Update
		LL_DMA_EnableCTR3Update
		LL_DMA_IsEnabledCTR3Update
CSAR	SA	LL_DMA_ConfigAddresses
		LL_DMA_GetSrcAddress
		LL_DMA_SetSrcAddress
CSR	DTEF	LL_DMA_IsActiveFlag_DTE

Register	Field	Function
CSR	FIFOL	LL_DMA_GetFIFOLevel
	HTF	LL_DMA_IsActiveFlag_HT
	IDLEF	LL_DMA_IsActiveFlag_IDLE
	SUSPF	LL_DMA_IsActiveFlag_SUSP
	TCF	LL_DMA_IsActiveFlag_TC
	TOF	LL_DMA_IsActiveFlag_TO
	ULEF	LL_DMA_IsActiveFlag_ULE
	USEF	LL_DMA_IsActiveFlag_USE
CTR1	DAP	LL_DMA_ConfigTransfer
		LL_DMA_GetDestAllocatedPort
		LL_DMA_SetDestAllocatedPort
	DBL_1	LL_DMA_GetDestBurstLength
		LL_DMA_SetDestBurstLength
	DBX	LL_DMA_ConfigTransfer
		LL_DMA_GetDestByteExchange
		LL_DMA_SetDestByteExchange
	DDW_LOG2	LL_DMA_GetDestDataWidth
		LL_DMA_SetDestDataWidth
	DHX	LL_DMA_ConfigTransfer
		LL_DMA_GetDestHWordExchange
		LL_DMA_SetDestHWordExchange
	DINC	LL_DMA_ConfigTransfer
		LL_DMA_GetDestIncMode
		LL_DMA_SetDestIncMode
	DSEC	LL_DMA_ConfigChannelSecure
		LL_DMA_DisableChannelDestSecure
		LL_DMA_EnableChannelDestSecure
		LL_DMA_IsEnabledChannelDestSecure
	PAM	LL_DMA_ConfigTransfer
		LL_DMA_GetDataAlignment
		LL_DMA_SetDataAlignment
	SAP	LL_DMA_ConfigTransfer
		LL_DMA_GetSrcAllocatedPort
		LL_DMA_SetSrcAllocatedPort
	SBL_1	LL_DMA_GetSrcBurstLength
		LL_DMA_SetDestBurstLength
		LL_DMA_SetSrcBurstLength
	SBX	LL_DMA_ConfigTransfer
		LL_DMA_GetSrcByteExchange
		LL_DMA_SetSrcByteExchange
	SDW_LOG2	LL_DMA_GetSrcDataWidth

Register	Field	Function
CTR1	SDW_LOG2	LL_DMA_SetSrcDataWidth
	SINC	LL_DMA_ConfigTransfer
		LL_DMA_GetSrcIncMode
		LL_DMA_SetSrcIncMode
	SSEC	LL_DMA_ConfigChannelSecure
		LL_DMA_DisableChannelSrcSecure
		LL_DMA_EnableChannelSrcSecure
		LL_DMA_IsEnabledChannelSrcSecure
CTR2	BREQ	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetBlkHWRequest
		LL_DMA_SetBlkHWRequest
	DREQ	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	REQSEL	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
	SWREQ	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	TCEM	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetTransferEventMode
		LL_DMA_SetTransferEventMode
	TRIGM	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetTriggerMode
		LL_DMA_SetTriggerMode
	TRIGPOL	LL_DMA_ConfigChannelTransfer
		LL_DMA_GetTriggerPolarity
		LL_DMA_SetTriggerPolarity
	TRIGSEL	LL_DMA_GetHWTrigger
		LL_DMA_SetHWTrigger
CTR3	DAO	LL_DMA_ConfigAddrUpdateValue
		LL_DMA_SetDestAddrUpdateValue
	SAO	LL_DMA_ConfigAddrUpdateValue
		LL_DMA_GetSrcAddrUpdateValue
		LL_DMA_SetSrcAddrUpdateValue
MISR	MISx	LL_DMA_IsActiveFlag_MIS
PRIVCFGR	PRIVx	LL_DMA_DisableChannelPrivilege
		LL_DMA_EnableChannelPrivilege
		LL_DMA_IsEnabledChannelPrivilege
RCFGLOCKR	LOCKx	LL_DMA_EnableChannelLockAttribute
SECCFGR	LOCKx	LL_DMA_IsEnabledChannelLockAttribute

Register	Field	Function
SECCFGR	SEC	LL_DMA_ConfigChannelSecure
	SECx	LL_DMA_DisableChannelSecure
		LL_DMA_EnableChannelSecure
		LL_DMA_IsEnabledChannelSecure
SMISR	MISx	LL_DMA_IsActiveFlag_SMIS

124.11 DMA2D

Table 35. **Correspondence between DMA2D registers and DMA2D low-layer driver functions**

Register	Field	Function
AMTCR	DT	LL_DMA2D_GetDeadTime
		LL_DMA2D_SetDeadTime
	EN	LL_DMA2D_DisableDeadTime
		LL_DMA2D_EnableDeadTime
		LL_DMA2D_IsEnabledDeadTime
BGCMAR	MA	LL_DMA2D_BGND_GetCLUTMemAddr
		LL_DMA2D_BGND_SetCLUTMemAddr
BGCOLR	BLUE	LL_DMA2D_BGND_GetBlueColor
		LL_DMA2D_BGND_SetBlueColor
		LL_DMA2D_BGND_SetColor
	GREEN	LL_DMA2D_BGND_GetGreenColor
		LL_DMA2D_BGND_SetColor
		LL_DMA2D_BGND_SetGreenColor
	RED	LL_DMA2D_BGND_GetRedColor
		LL_DMA2D_BGND_SetColor
		LL_DMA2D_BGND_SetRedColor
BGMAR	MA	LL_DMA2D_BGND_GetMemAddr
		LL_DMA2D_BGND_SetMemAddr
BGOR	LO	LL_DMA2D_BGND_GetLineOffset
		LL_DMA2D_BGND_SetLineOffset
BGPFCCR	AI	LL_DMA2D_BGND_GetAlphaInvMode
		LL_DMA2D_BGND_SetAlphaInvMode
	ALPHA	LL_DMA2D_BGND_GetAlpha
		LL_DMA2D_BGND_SetAlpha
	AM	LL_DMA2D_BGND_GetAlphaMode
		LL_DMA2D_BGND_SetAlphaMode
	CCM	LL_DMA2D_BGND_GetCLUTColorMode
		LL_DMA2D_BGND_SetCLUTColorMode
	CM	LL_DMA2D_BGND_GetColorMode
		LL_DMA2D_BGND_SetColorMode
	CS	LL_DMA2D_BGND_GetCLUTSize

Register	Field	Function
BGPFCCR	CS	LL_DMA2D_BGND_SetCLUTSize
	RBS	LL_DMA2D_BGND_GetRBSwapMode
		LL_DMA2D_BGND_SetRBSwapMode
	START	LL_DMA2D_BGND_EnableCLUTLoad
		LL_DMA2D_BGND_IsEnabledCLUTLoad
CR	ABORT	LL_DMA2D_Abort
		LL_DMA2D_IsAborted
	CAEIE	LL_DMA2D_DisableIT_CAE
		LL_DMA2D_EnableIT_CAE
		LL_DMA2D_IsEnabledIT_CAE
	CEIE	LL_DMA2D_DisableIT_CE
		LL_DMA2D_EnableIT_CE
		LL_DMA2D_IsEnabledIT_CE
	CTCIE	LL_DMA2D_DisableIT_CTC
		LL_DMA2D_EnableIT_CTC
		LL_DMA2D_IsEnabledIT_CTC
	LOM	LL_DMA2D_GetLineOffsetMode
		LL_DMA2D_SetLineOffsetMode
	MODE	LL_DMA2D_GetMode
		LL_DMA2D_SetMode
	START	LL_DMA2D_IsTransferOngoing
		LL_DMA2D_Start
	SUSP	LL_DMA2D_IsSuspended
		LL_DMA2D_Resume
		LL_DMA2D_Suspend
	TCIE	LL_DMA2D_DisableIT_TC
		LL_DMA2D_EnableIT_TC
		LL_DMA2D_IsEnabledIT_TC
	TEIE	LL_DMA2D_DisableIT_TE
		LL_DMA2D_EnableIT_TE
		LL_DMA2D_IsEnabledIT_TE
	TWIE	LL_DMA2D_DisableIT_TW
		LL_DMA2D_EnableIT_TW
		LL_DMA2D_IsEnabledIT_TW
FGCMAR	MA	LL_DMA2D_FGND_GetCLUTMemAddr
		LL_DMA2D_FGND_SetCLUTMemAddr
FGCOLR	BLUE	LL_DMA2D_FGND_GetBlueColor
		LL_DMA2D_FGND_SetBlueColor
		LL_DMA2D_FGND_SetColor
	GREEN	LL_DMA2D_FGND_GetGreenColor
		LL_DMA2D_FGND_SetColor

Register	Field	Function
FGCOLR	GREEN	LL_DMA2D_FGND_SetGreenColor
	RED	LL_DMA2D_FGND_GetRedColor
		LL_DMA2D_FGND_SetColor
		LL_DMA2D_FGND_SetRedColor
FGMAR	MA	LL_DMA2D_FGND_GetMemAddr
		LL_DMA2D_FGND_SetMemAddr
FGOR	LO	LL_DMA2D_FGND_GetLineOffset
		LL_DMA2D_FGND_SetLineOffset
FGPFCCR	AI	LL_DMA2D_FGND_GetAlphaInvMode
		LL_DMA2D_FGND_SetAlphaInvMode
	ALPHA	LL_DMA2D_FGND_GetAlpha
		LL_DMA2D_FGND_SetAlpha
	AM	LL_DMA2D_FGND_GetAlphaMode
		LL_DMA2D_FGND_SetAlphaMode
	CCM	LL_DMA2D_FGND_GetCLUTColorMode
		LL_DMA2D_FGND_SetCLUTColorMode
	CM	LL_DMA2D_FGND_GetColorMode
		LL_DMA2D_FGND_SetColorMode
	CS	LL_DMA2D_FGND_GetCLUTSize
		LL_DMA2D_FGND_SetCLUTSize
	CSS	LL_DMA2D_FGND_GetChrSubSampling
		LL_DMA2D_FGND_SetChrSubSampling
	RBS	LL_DMA2D_FGND_GetRBSwapMode
		LL_DMA2D_FGND_SetRBSwapMode
	START	LL_DMA2D_FGND_EnableCLUTLoad
		LL_DMA2D_FGND_IsEnabledCLUTLoad
IFCR	CAECIF	LL_DMA2D_ClearFlag_CAE
	CCEIF	LL_DMA2D_ClearFlag_CE
	CCTCIF	LL_DMA2D_ClearFlag_CTC
	CTCIF	LL_DMA2D_ClearFlag_TC
	CTEIF	LL_DMA2D_ClearFlag_TE
	CTWIF	LL_DMA2D_ClearFlag_TW
ISR	CAEIF	LL_DMA2D_IsActiveFlag_CAE
	CEIF	LL_DMA2D_IsActiveFlag_CE
	CTCIF	LL_DMA2D_IsActiveFlag_CTC
	TCIF	LL_DMA2D_IsActiveFlag_TC
	TEIF	LL_DMA2D_IsActiveFlag_TE
	TWIF	LL_DMA2D_IsActiveFlag_TW
LWR	LW	LL_DMA2D_GetLineWatermark
		LL_DMA2D_SetLineWatermark
NLR	NL	LL_DMA2D_GetNbrOfLines

Register	Field	Function
NLR	NL	LL_DMA2D_SetNbrOfLines
	PL	LL_DMA2D_GetNbrOfPixelsPerLines
		LL_DMA2D_SetNbrOfPixelsPerLines
OCOLR	ALPHA	LL_DMA2D_GetOutputColor
		LL_DMA2D_SetOutputColor
	BLUE	LL_DMA2D_GetOutputColor
		LL_DMA2D_SetOutputColor
	GREEN	LL_DMA2D_GetOutputColor
		LL_DMA2D_SetOutputColor
	RED	LL_DMA2D_GetOutputColor
		LL_DMA2D_SetOutputColor
OMAR	MA	LL_DMA2D_GetOutputMemAddr
		LL_DMA2D_SetOutputMemAddr
OOR	LO	LL_DMA2D_GetLineOffset
		LL_DMA2D_SetLineOffset
OPFCCR	AI	LL_DMA2D_GetOutputAlphaInvMode
		LL_DMA2D_SetOutputAlphaInvMode
	CM	LL_DMA2D_GetOutputColorMode
		LL_DMA2D_SetOutputColorMode
	RBS	LL_DMA2D_GetOutputRBSwapMode
		LL_DMA2D_SetOutputRBSwapMode
	SB	LL_DMA2D_GetOutputSwapMode
		LL_DMA2D_SetOutputSwapMode

124.12 EXTI

Table 36. **Correspondence between EXTI registers and EXTI low-layer driver functions**

Register	Field	Function
EMR1	EMx	LL_EXTI_DisableEvent_0_31
		LL_EXTI_EnableEvent_0_31
		LL_EXTI_IsEnabledEvent_0_31
EXTI_EXTICR1	EXTI0	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI1	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI2	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI3	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
EXTI_EXTICR2	EXTI4	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource

Register	Field	Function
EXTI_EXTICR2	EXTI5	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI6	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI7	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
EXTI_EXTICR3	EXTI10	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI11	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI8	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI9	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
EXTI_EXTICR4	EXTI12	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI13	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI14	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
	EXTI15	LL_EXTI_GetEXTISource
		LL_EXTI_SetEXTISource
FPR1	FPIFx	LL_EXTI_ClearFallingFlag_0_31
		LL_EXTI_IsActiveFallingFlag_0_31
		LL_EXTI_ReadFallingFlag_0_31
FTSR1	FTx	LL_EXTI_DisableFallingTrig_0_31
		LL_EXTI_EnableFallingTrig_0_31
		LL_EXTI_IsEnabledFallingTrig_0_31
IMR1	IMx	LL_EXTI_DisableIT_0_31
		LL_EXTI_EnableIT_0_31
		LL_EXTI_IsEnabledIT_0_31
LOCKR	LOCK	LL_EXTI_GetLockAttributes
		LL_EXTI_LockAttributes
PRIVCFGR1	PRIVx	LL_EXTI_DisablePrivilege_0_31
		LL_EXTI_EnablePrivilege_0_31
		LL_EXTI_IsEnabledPrivilege_0_31
RPR1	RPIFx	LL_EXTI_ClearRisingFlag_0_31
		LL_EXTI_IsActiveRisingFlag_0_31
		LL_EXTI_ReadRisingFlag_0_31
RTSR1	RTx	LL_EXTI_DisableRisingTrig_0_31
		LL_EXTI_EnableRisingTrig_0_31

Register	Field	Function
RTSR1	RTx	LL_EXTI_IsEnabledRisingTrig_0_31
SECCFGR1	SECx	LL_EXTI_DisableSecure_0_31
		LL_EXTI_EnableSecure_0_31
		LL_EXTI_IsEnabledSecure_0_31
SWIER1	SWIx	LL_EXTI_GenerateSWI_0_31

124.13 FMAC

Table 37. Correspondence between FMAC registers and FMAC low-layer driver functions

Register	Field	Function
CR	CLIPEN	LL_FMAC_DisableClipping
		LL_FMAC_EnableClipping
		LL_FMAC_IsEnabledClipping
	DMAREN	LL_FMAC_DisableDMAReq_READ
		LL_FMAC_EnableDMAReq_READ
		LL_FMAC_IsEnabledDMAReq_READ
	DMAWEN	LL_FMAC_DisableDMAReq_WRITE
		LL_FMAC_EnableDMAReq_WRITE
		LL_FMAC_IsEnabledDMAReq_WRITE
	OVFLIEN	LL_FMAC_DisableIT_OVFL
		LL_FMAC_EnableIT_OVFL
		LL_FMAC_IsEnabledIT_OVFL
	RESET	LL_FMAC_EnableReset
		LL_FMAC_IsEnabledReset
	RIEN	LL_FMAC_DisableIT_RD
		LL_FMAC_EnableIT_RD
		LL_FMAC_IsEnabledIT_RD
	SATIEN	LL_FMAC_DisableIT_SAT
		LL_FMAC_EnableIT_SAT
		LL_FMAC_IsEnabledIT_SAT
	UNFLIEN	LL_FMAC_DisableIT_UNFL
		LL_FMAC_EnableIT_UNFL
		LL_FMAC_IsEnabledIT_UNFL
	WIEN	LL_FMAC_DisableIT_WR
		LL_FMAC_EnableIT_WR
		LL_FMAC_IsEnabledIT_WR
PARAM	FUNC	LL_FMAC_ConfigFunc
		LL_FMAC_GetFunction
		LL_FMAC_SetFunction
	P	LL_FMAC_ConfigFunc
		LL_FMAC_GetParamP

Register	Field	Function
PARAM	P	LL_FMAC_SetParamP
	Q	LL_FMAC_ConfigFunc
		LL_FMAC_GetParamQ
		LL_FMAC_SetParamQ
	R	LL_FMAC_ConfigFunc
		LL_FMAC_GetParamR
		LL_FMAC_SetParamR
	START	LL_FMAC_ConfigFunc
		LL_FMAC_DisableStart
		LL_FMAC_EnableStart
		LL_FMAC_IsEnabledStart
RDATA	RDATA	LL_FMAC_ReadData
SR	OVFL	LL_FMAC_IsActiveFlag_OVFL
	SAT	LL_FMAC_IsActiveFlag_SAT
	UNFL	LL_FMAC_IsActiveFlag_UNFL
	X1FULL	LL_FMAC_IsActiveFlag_X1FULL
	YEMPTY	LL_FMAC_IsActiveFlag_YEMPTY
WDATA	WDATA	LL_FMAC_WriteData
X1BUFCFG	FULL_WM	LL_FMAC_ConfigX1
		LL_FMAC_GetX1FullWatermark
		LL_FMAC_SetX1FullWatermark
	X1_BASE	LL_FMAC_ConfigX1
		LL_FMAC_GetX1Base
		LL_FMAC_SetX1Base
	X1_BUF_SIZE	LL_FMAC_ConfigX1
		LL_FMAC_GetX1BufferSize
		LL_FMAC_SetX1BufferSize
X2BUFCFG	X2_BASE	LL_FMAC_ConfigX2
		LL_FMAC_GetX2Base
		LL_FMAC_SetX2Base
	X2_BUF_SIZE	LL_FMAC_ConfigX2
		LL_FMAC_GetX2BufferSize
		LL_FMAC_SetX2BufferSize
YBUFCFG	EMPTY_WM	LL_FMAC_ConfigY
		LL_FMAC_GetYEmptyWatermark
		LL_FMAC_SetYEmptyWatermark
	Y_BASE	LL_FMAC_ConfigY
		LL_FMAC_GetYBase
		LL_FMAC_SetYBase
	Y_BUF_SIZE	LL_FMAC_ConfigY
		LL_FMAC_GetYBufferSize

Register	Field	Function
YBUFCFG	Y_BUF_SIZE	LL_FMAC_SetYBufferSize

124.14 GPIO

Table 38. Correspondence between GPIO registers and GPIO low-layer driver functions

Register	Field	Function
AFRH	AFSELy	LL_GPIO_GetAFPin_8_15
		LL_GPIO_SetAFPin_8_15
AFRL	AFSELy	LL_GPIO_GetAFPin_0_7
		LL_GPIO_SetAFPin_0_7
BRR	BRy	LL_GPIO_ResetOutputPin
BSRR	BSy	LL_GPIO_SetOutputPin
HSLVR	HSLVy	LL_GPIO_DisableHighSPeedLowVoltage
		LL_GPIO_EnableHighSPeedLowVoltage
		LL_GPIO_IsEnabledHighSPeedLowVoltage
IDR	IDy	LL_GPIO_IsInputPinSet
		LL_GPIO_ReadInputPort
LCKR	LCKK	LL_GPIO_IsAnyPinLocked
		LL_GPIO_LockPin
	LCKy	LL_GPIO_IsPinLocked
MODER	MODEy	LL_GPIO_GetPinMode
		LL_GPIO_SetPinMode
ODR	ODy	LL_GPIO_IsOutputPinSet
		LL_GPIO_ReadOutputPort
		LL_GPIO_TogglePin
		LL_GPIO_WriteOutputPort
OSPEEDR	OSPEEDy	LL_GPIO_GetPinSpeed
		LL_GPIO_SetPinSpeed
OTYPER	OTy	LL_GPIO_GetPinOutputType
		LL_GPIO_SetPinOutputType
PUPDR	PUPDy	LL_GPIO_GetPinPull
		LL_GPIO_SetPinPull
SECCFGR	SECy	LL_GPIO_DisablePinSecure
		LL_GPIO_EnablePinSecure
		LL_GPIO_IsEnabledPinSecure

124.15 I2C

Table 39. Correspondence between I2C registers and I2C low-layer driver functions

Register	Field	Function
AUTOCR	TRIGEN	LL_I2C_Disable_SelectedTrigger

Register	Field	Function
AUTOCR	TRIGEN	LL_I2C_Enable_SelectedTrigger
		LL_I2C_IsEnabled_SelectedTrigger
	TRIGPOL	LL_I2C_GetTriggerPolarity
		LL_I2C_SetTriggerPolarity
	TRIGSEL	LL_I2C_GetSelectedTrigger
		LL_I2C_SetSelectedTrigger
CR1	ADDRACLR	LL_I2C_DisableAutoClearFlag_ADDR
		LL_I2C_EnableAutoClearFlag_ADDR
		LL_I2C_IsEnabledAutoClearFlag_ADDR
	ADDRIE	LL_I2C_DisableIT_ADDR
		LL_I2C_EnableIT_ADDR
		LL_I2C_IsEnabledIT_ADDR
	ALERTEN	LL_I2C_DisableSMBusAlert
		LL_I2C_EnableSMBusAlert
		LL_I2C_IsEnabledSMBusAlert
	ANFOFF	LL_I2C_ConfigFilters
		LL_I2C_DisableAnalogFilter
		LL_I2C_EnableAnalogFilter
		LL_I2C_IsEnabledAnalogFilter
	DNF	LL_I2C_ConfigFilters
		LL_I2C_GetDigitalFilter
		LL_I2C_SetDigitalFilter
	ERRIE	LL_I2C_DisableIT_ERR
		LL_I2C_EnableIT_ERR
		LL_I2C_IsEnabledIT_ERR
	FMP	LL_I2C_DisableFastModePlus
		LL_I2C_EnableFastModePlus
		LL_I2C_IsEnabledFastModePlus
	GCEN	LL_I2C_DisableGeneralCall
		LL_I2C_EnableGeneralCall
		LL_I2C_IsEnabledGeneralCall
	NACKIE	LL_I2C_DisableIT_NACK
		LL_I2C_EnableIT_NACK
		LL_I2C_IsEnabledIT_NACK
	NOSTRETCH	LL_I2C_DisableClockStretching
		LL_I2C_EnableClockStretching
		LL_I2C_IsEnabledClockStretching
	PE	LL_I2C_Disable
		LL_I2C_Enable
		LL_I2C_IsEnabled
	PECEN	LL_I2C_DisableSMBusPEC

Register	Field	Function
CR1	PECEN	LL_I2C_EnableSMBusPEC
		LL_I2C_IsEnabledSMBusPEC
	RXDMAEN	LL_I2C_DisableDMAReq_RX
		LL_I2C_EnableDMAReq_RX
		LL_I2C_IsEnabledDMAReq_RX
	RXIE	LL_I2C_DisableIT_RX
		LL_I2C_EnableIT_RX
		LL_I2C_IsEnabledIT_RX
	SBC	LL_I2C_DisableSlaveByteControl
		LL_I2C_EnableSlaveByteControl
		LL_I2C_IsEnabledSlaveByteControl
	SMBDEN	LL_I2C_GetMode
		LL_I2C_SetMode
	SMBHEN	LL_I2C_GetMode
		LL_I2C_SetMode
	STOPFACLR	LL_I2C_DisableAutoClearFlag_STOP
		LL_I2C_EnableAutoClearFlag_STOP
		LL_I2C_IsEnabledAutoClearFlag_STOP
	STOPIE	LL_I2C_DisableIT_STOP
		LL_I2C_EnableIT_STOP
		LL_I2C_IsEnabledIT_STOP
	TCIE	LL_I2C_DisableIT_TC
		LL_I2C_EnableIT_TC
		LL_I2C_IsEnabledIT_TC
	TXDMAEN	LL_I2C_DisableDMAReq_TX
		LL_I2C_EnableDMAReq_TX
		LL_I2C_IsEnabledDMAReq_TX
	TXIE	LL_I2C_DisableIT_TX
		LL_I2C_EnableIT_TX
		LL_I2C_IsEnabledIT_TX
	WUPEN	LL_I2C_DisableWakeUpFromStop
		LL_I2C_EnableWakeUpFromStop
		LL_I2C_IsEnabledWakeUpFromStop
CR2	ADD10	LL_I2C_GetMasterAddressingMode
		LL_I2C_HandleTransfer
		LL_I2C_SetMasterAddressingMode
	AUTOEND	LL_I2C_DisableAutoEndMode
		LL_I2C_EnableAutoEndMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAutoEndMode
	HEAD10R	LL_I2C_DisableAuto10BitRead

Register	Field	Function
CR2	HEAD10R	LL_I2C_EnableAuto10BitRead
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAuto10BitRead
	NACK	LL_I2C_AcknowledgeNextData
	NBYTES	LL_I2C_GetTransferSize
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferSize
	PECBYTE	LL_I2C_EnableSMBusPECCompare
		LL_I2C_IsEnabledSMBusPECCompare
	RD_WRN	LL_I2C_GetTransferRequest
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferRequest
	RELOAD	LL_I2C_DisableReloadMode
		LL_I2C_EnableReloadMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledReloadMode
	SADD	LL_I2C_GetSlaveAddr
		LL_I2C_HandleTransfer
		LL_I2C_SetSlaveAddr
	START	LL_I2C_GenerateStartCondition
		LL_I2C_HandleTransfer
	STOP	LL_I2C_GenerateStopCondition
		LL_I2C_HandleTransfer
ICR	ADDRCF	LL_I2C_ClearFlag_ADDR
	ALERTCF	LL_I2C_ClearSMBusFlag_ALERT
	ARLOCF	LL_I2C_ClearFlag_ARLO
	BERRCF	LL_I2C_ClearFlag_BERR
	NACKCF	LL_I2C_ClearFlag_NACK
	OVRCF	LL_I2C_ClearFlag_OVR
	PECCF	LL_I2C_ClearSMBusFlag_PECERR
	STOPCF	LL_I2C_ClearFlag_STOP
	TIMOUTCF	LL_I2C_ClearSMBusFlag_TIMEOUT
ISR	ADDCODE	LL_I2C_GetAddressMatchCode
	ADDR	LL_I2C_IsActiveFlag_ADDR
	ALERT	LL_I2C_IsActiveSMBusFlag_ALERT
	ARLO	LL_I2C_IsActiveFlag_ARLO
	BERR	LL_I2C_IsActiveFlag_BERR
	BUSY	LL_I2C_IsActiveFlag_BUSY
	DIR	LL_I2C_GetTransferDirection
	NACKF	LL_I2C_IsActiveFlag_NACK
	OVR	LL_I2C_IsActiveFlag_OVR

Register	Field	Function
ISR	PECERR	LL_I2C_IsActiveSMBusFlag_PECERR
	RXNE	LL_I2C_IsActiveFlag_RXNE
	STOPF	LL_I2C_IsActiveFlag_STOP
	TC	LL_I2C_IsActiveFlag_TC
	TCR	LL_I2C_IsActiveFlag_TCR
	TIMEOUT	LL_I2C_IsActiveSMBusFlag_TIMEOUT
	TXE	LL_I2C_ClearFlag_TXE
		LL_I2C_IsActiveFlag_TXE
	TXIS	LL_I2C_IsActiveFlag_TXIS
OAR1	OA1	LL_I2C_SetOwnAddress1
	OA1EN	LL_I2C_DisableOwnAddress1
		LL_I2C_EnableOwnAddress1
		LL_I2C_IsEnabledOwnAddress1
	OA1MODE	LL_I2C_SetOwnAddress1
OAR2	OA2	LL_I2C_SetOwnAddress2
	OA2EN	LL_I2C_DisableOwnAddress2
		LL_I2C_EnableOwnAddress2
		LL_I2C_IsEnabledOwnAddress2
	OA2MSK	LL_I2C_SetOwnAddress2
PECR	PEC	LL_I2C_GetSMBusPEC
RXDR	RXDATA	LL_I2C_DMA_GetRegAddr
		LL_I2C_ReceiveData8
TIMEOUTR	TEXTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
	TIDLE	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutAMode
		LL_I2C_SetSMBusTimeoutAMode
	TIMEOUTA	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutA
		LL_I2C_SetSMBusTimeoutA
	TIMEOUTB	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutB
		LL_I2C_SetSMBusTimeoutB
	TIMOUTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
TIMINGR	PRESC	LL_I2C_GetTimingPrescaler
	SCLDEL	LL_I2C_GetDataSetupTime
	SCLH	LL_I2C_GetClockHighPeriod
	SCLL	LL_I2C_GetClockLowPeriod

Register	Field	Function
TIMINGR	SDADEL	LL_I2C_GetDataHoldTime
	TIMINGR	LL_I2C_SetTiming
TXDR	TXDATA	LL_I2C_DMA_GetRegAddr
		LL_I2C_TransmitData8

124.16 ICACHE

Table 40. Correspondence between ICACHE registers and ICACHE low-layer driver functions

Register	Field	Function
CR	CACHEINV	LL_ICACHE_Invalidate
	EN	LL_ICACHE_Disable
		LL_ICACHE_Enable
		LL_ICACHE_IsEnabled
	HITMEN	LL_ICACHE_DisableMonitors
		LL_ICACHE_EnableMonitors
		LL_ICACHE_IsEnabledMonitors
	HITMRST	LL_ICACHE_ResetMonitors
	MISSMEN	LL_ICACHE_DisableMonitors
		LL_ICACHE_EnableMonitors
		LL_ICACHE_IsEnabledMonitors
	MISSMRST	LL_ICACHE_ResetMonitors
	WAYSEL	LL_ICACHE_GetMode
		LL_ICACHE_SetMode
CRRx	BASEADDR	LL_ICACHE_GetRegionBaseAddress
		LL_ICACHE_SetRegionBaseAddress
	HBURST	LL_ICACHE_GetRegionOutputBurstType
		LL_ICACHE_SetRegionOutputBurstType
	MSTSEL	LL_ICACHE_GetRegionMasterPort
		LL_ICACHE_SetRegionMasterPort
	REMAPADDR	LL_ICACHE_GetRegionRemapAddress
		LL_ICACHE_SetRegionRemapAddress
	REN	LL_ICACHE_DisableRegion
		LL_ICACHE_EnableRegion
		LL_ICACHE_IsEnabledRegion
	RSIZE	LL_ICACHE_GetRegionSize
		LL_ICACHE_SetRegionSize
FCR	CBSYENDF	LL_ICACHE_ClearFlag_BSYEND
	ERRF	LL_ICACHE_ClearFlag_ERR
HMONR	HITMON	LL_ICACHE_GetHitMonitor
IER	BSYENDIE	LL_ICACHE_DisableIT_BSYEND
		LL_ICACHE_EnableIT_BSYEND

Register	Field	Function
IER	BSYENDIE	LL_ICACHE_IsEnabledIT_BSYEND
	ERRIE	LL_ICACHE_DisableIT_ERR
		LL_ICACHE_EnableIT_ERR
		LL_ICACHE_IsEnabledIT_ERR
MMONR	MISSMON	LL_ICACHE_GetMissMonitor
SR	BSYEND	LL_ICACHE_IsActiveFlag_BSYEND
	BUSYF	LL_ICACHE_IsActiveFlag_BUSY
	ERRF	LL_ICACHE_IsActiveFlag_ERR

124.17 IWDG

Table 41. Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
EWCR	EWIC	LL_IWDG_ClearFlag_EWIF
	EWIE	LL_IWDG_DisableIT_EWI
		LL_IWDG_EnableIT_EWI
		LL_IWDG_IsEnabledIT_EWI
	EWIT	LL_IWDG_GetEwiTime
		LL_IWDG_SetEwiTime
KR	KEY	LL_IWDG_DisableWriteAccess
		LL_IWDG_Enable
		LL_IWDG_EnableWriteAccess
		LL_IWDG_ReloadCounter
PR	PR	LL_IWDG_GetPrescaler
		LL_IWDG_SetPrescaler
RLR	RL	LL_IWDG_GetReloadCounter
		LL_IWDG_SetReloadCounter
SR	EVU	LL_IWDG_IsActiveFlag_EWU
	EWIF	LL_IWDG_IsActiveFlag_EWIF
	EWU	LL_IWDG_IsReady
	PVU	LL_IWDG_IsActiveFlag_PVU
		LL_IWDG_IsReady
	RVU	LL_IWDG_IsActiveFlag_RVU
		LL_IWDG_IsReady
	WVU	LL_IWDG_IsActiveFlag_WVU
		LL_IWDG_IsReady
WINR	WIN	LL_IWDG_GetWindow
		LL_IWDG_SetWindow

124.18 LPGPIO

Table 42. Correspondence between LPGPIO registers and LPGPIO low-layer driver functions

Register	Field	Function
BRR	BRy	LL_LPGPIO_ResetOutputPin
BSRR	BSy	LL_LPGPIO_SetOutputPin
IDR	IDy	LL_LPGPIO_IsInputPinSet
		LL_LPGPIO_ReadInputPort
MODER	MODEy	LL_LPGPIO_GetPinMode
		LL_LPGPIO_SetPinMode
ODR	ODy	LL_LPGPIO_IsOutputPinSet
		LL_LPGPIO_ReadOutputPort
		LL_LPGPIO_TogglePin
		LL_LPGPIO_WriteOutputPort

124.19 LPTIM

Table 43. Correspondence between LPTIM registers and LPTIM low-layer driver functions

Register	Field	Function
ARR	ARR	LL_LPTIM_GetAutoReload
		LL_LPTIM_SetAutoReload
CCMR1	CC1E	LL_LPTIM_CC_DisableChannel
		LL_LPTIM_CC_EnableChannel
		LL_LPTIM_CC_IsEnabledChannel
	CC1P	LL_LPTIM_IC_GetPolarity
		LL_LPTIM_IC_SetPolarity
		LL_LPTIM_OC_GetPolarity
		LL_LPTIM_OC_SetPolarity
	CC1SEL	LL_LPTIM_CC_GetChannelMode
		LL_LPTIM_CC_SetChannelMode
	CC2E	LL_LPTIM_CC_DisableChannel
		LL_LPTIM_CC_EnableChannel
		LL_LPTIM_CC_IsEnabledChannel
	CC2P	LL_LPTIM_IC_GetPolarity
		LL_LPTIM_IC_SetPolarity
		LL_LPTIM_OC_GetPolarity
		LL_LPTIM_OC_SetPolarity
	CC2SEL	LL_LPTIM_CC_GetChannelMode
		LL_LPTIM_CC_SetChannelMode
	IC1F	LL_LPTIM_IC_GetFilter
		LL_LPTIM_IC_SetFilter
	IC1PSC	LL_LPTIM_IC_GetPrescaler

Register	Field	Function
CCMR1	IC1PSC	LL_LPTIM_IC_SetPrescaler
	IC2F	LL_LPTIM_IC_GetFilter
		LL_LPTIM_IC_SetFilter
	IC2PSC	LL_LPTIM_IC_GetPrescaler
		LL_LPTIM_IC_SetPrescaler
CCR1	CCR1	LL_LPTIM_IC_GetCaptureCH1
		LL_LPTIM_OC_GetCompareCH1
		LL_LPTIM_OC_SetCompareCH1
CCR2	CCR2	LL_LPTIM_IC_GetCaptureCH2
		LL_LPTIM_OC_GetCompareCH2
		LL_LPTIM_OC_SetCompareCH2
CFGR	CKFLT	LL_LPTIM_ConfigClock
		LL_LPTIM_GetClockFilter
	CKPOL	LL_LPTIM_ConfigClock
		LL_LPTIM_GetClockPolarity
		LL_LPTIM_GetEncoderMode
		LL_LPTIM_SetEncoderMode
	CKSEL	LL_LPTIM_GetClockSource
		LL_LPTIM_SetClockSource
	COUNTMODE	LL_LPTIM_GetCounterMode
		LL_LPTIM_SetCounterMode
	ENC	LL_LPTIM_DisableEncoderMode
		LL_LPTIM_EnableEncoderMode
		LL_LPTIM_IsEnabledEncoderMode
	PRELOAD	LL_LPTIM_GetUpdateMode
		LL_LPTIM_SetUpdateMode
	PRESC	LL_LPTIM_GetPrescaler
		LL_LPTIM_SetPrescaler
	TIMOUT	LL_LPTIM_DisableTimeout
		LL_LPTIM_EnableTimeout
		LL_LPTIM_IsEnabledTimeout
	TRGFLT	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerFilter
	TRIGEN	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerPolarity
		LL_LPTIM_TrigSw
	TRIGSEL	LL_LPTIM_ConfigTrigger
		LL_LPTIM_GetTriggerSource
	WAVE	LL_LPTIM_GetWaveform
		LL_LPTIM_SetWaveform
CFGR2	IC1SEL	LL_LPTIM_SetRemap

Register	Field	Function
CFGR2	IC2SEL	LL_LPTIM_SetRemap
	IN1SEL	LL_LPTIM_SetInput1Src
	IN2SEL	LL_LPTIM_SetInput2Src
CNT	CNT	LL_LPTIM_GetCounter
CR	CNTSTRT	LL_LPTIM_StartCounter
	COUNTRST	LL_LPTIM_ResetCounter
	ENABLE	LL_LPTIM_Disable
		LL_LPTIM_Enable
		LL_LPTIM_IsEnabled
	RSTARE	LL_LPTIM_DisableResetAfterRead
		LL_LPTIM_EnableResetAfterRead
		LL_LPTIM_IsEnabledResetAfterRead
	SNGSTRT	LL_LPTIM_StartCounter
DIER	ARRMIE	LL_LPTIM_DisableIT_ARRM
		LL_LPTIM_EnableIT_ARRM
		LL_LPTIM_IsEnabledIT_ARRM
	ARROKIE	LL_LPTIM_DisableIT_ARROK
		LL_LPTIM_EnableIT_ARROK
		LL_LPTIM_IsEnabledIT_ARROK
	CC1IE	LL_LPTIM_DisableIT_CC1
		LL_LPTIM_EnableIT_CC1
		LL_LPTIM_IsEnabledIT_CC1
	CC1OIE	LL_LPTIM_DisableIT_CC1O
		LL_LPTIM_EnableIT_CC1O
		LL_LPTIM_IsEnabledIT_CC1O
	CC2IE	LL_LPTIM_DisableIT_CC2
		LL_LPTIM_EnableIT_CC2
		LL_LPTIM_IsEnabledIT_CC2
	CC2OIE	LL_LPTIM_DisableIT_CC2O
		LL_LPTIM_EnableIT_CC2O
		LL_LPTIM_IsEnabledIT_CC2O
	DOWNIE	LL_LPTIM_DisableIT_DOWN
		LL_LPTIM_EnableIT_DOWN
		LL_LPTIM_IsEnabledIT_DOWN
	EXTTRIGIE	LL_LPTIM_DisableIT_EXTTRIG
		LL_LPTIM_EnableIT_EXTTRIG
		LL_LPTIM_IsEnabledIT_EXTTRIG
	REPOKIE	LL_LPTIM_DisableIT_REPOK
		LL_LPTIM_EnableIT_REPOK
		LL_LPTIM_IsEnabledIT_REPOK
	UEIE	LL_LPTIM_DisableIT_UE

Register	Field	Function
DIER	UEIE	LL_LPTIM_EnableIT_UE
		LL_LPTIM_IsEnabledIT_UE
	UPIE	LL_LPTIM_DisableIT_UP
		LL_LPTIM_EnableIT_UP
		LL_LPTIM_IsEnabledIT_UP
ICR	ARRMCF	LL_LPTIM_ClearFlag_ARRM
	ARROKCF	LL_LPTIM_ClearFlag_ARROK
	CC1CF	LL_LPTIM_ClearFlag_CC1
	CC1OCF	LL_LPTIM_ClearFlag_CC1O
	CC2CF	LL_LPTIM_ClearFlag_CC2
	CC2OCF	LL_LPTIM_ClearFlag_CC2O
	CMP1OKCF	LL_LPTIM_ClearFlag_CMP1OK
	CMP2OKCF	LL_LPTIM_ClearFlag_CMP2OK
	DIEROKCF	LL_LPTIM_ClearFlag_DIEROK
	DOWNCF	LL_LPTIM_ClearFlag_DOWN
	EXTTRIGCF	LL_LPTIM_ClearFlag_EXTTRIG
	REPOKCF	LL_LPTIM_ClearFlag_REPOK
	UECF	LL_LPTIM_ClearFlag_UE
	UPCF	LL_LPTIM_ClearFlag_UP
IER	CMP1OKIE	LL_LPTIM_EnableIT_CMP1OK
		LL_LPTIM_IsEnabledIT_CMP1OK
	CMP2OKIE	LL_LPTIM_DisableIT_CMP2OK
		LL_LPTIM_EnableIT_CMP2OK
		LL_LPTIM_IsEnabledIT_CMP2OK
	CMPO1KIE	LL_LPTIM_DisableIT_CMP1OK
ISR	ARRM	LL_LPTIM_IsActiveFlag_ARRM
	ARROK	LL_LPTIM_IsActiveFlag_ARROK
	CC1IF	LL_LPTIM_IsActiveFlag_CC1
	CC1OF	LL_LPTIM_IsActiveFlag_CC1O
	CC2IF	LL_LPTIM_IsActiveFlag_CC2
	CC2OF	LL_LPTIM_IsActiveFlag_CC2O
	CMP1OK	LL_LPTIM_IsActiveFlag_CMP1OK
	CMP2OK	LL_LPTIM_IsActiveFlag_CMP2OK
	DIEROK	LL_LPTIM_IsActiveFlag_DIEROK
	DOWN	LL_LPTIM_IsActiveFlag_DOWN
	EXTTRIG	LL_LPTIM_IsActiveFlag_EXTTRIG
	REPOK	LL_LPTIM_IsActiveFlag_REPOK
	UE	LL_LPTIM_IsActiveFlag_UE
	UP	LL_LPTIM_IsActiveFlag_UP
RCR	REP	LL_LPTIM_GetRepetition
		LL_LPTIM_SetRepetition

124.20 LPUART

Table 44. Correspondence between LPUART registers and LPUART low-layer driver functions

Register	Field	Function
AUTOCR	IDLEDIS	LL_LPUART_Disable_AutonomousSendIdleFrame
		LL_LPUART_Enable_AutonomousSendIdleFrame
		LL_LPUART_IsEnabled_AutonomousSendIdleFrame
	TDN	LL_LPUART_GetNbTxData
		LL_LPUART_SetNbTxData
	TRIGEN	LL_LPUART_Disable_SelectedTrigger
		LL_LPUART_Enable_SelectedTrigger
		LL_LPUART_IsEnabled_SelectedTrigger
	TRIGPOL	LL_LPUART_GetTriggerPolarity
		LL_LPUART_SetTriggerPolarity
	TRIGSEL	LL_LPUART_GetSelectedTrigger
		LL_LPUART_SetSelectedTrigger
BRR	BRR	LL_LPUART_GetBaudRate
		LL_LPUART_SetBaudRate
CR1	CMIE	LL_LPUART_DisableIT_CM
		LL_LPUART_EnableIT_CM
		LL_LPUART_IsEnabledIT_CM
	DEAT	LL_LPUART_GetDEAssertionTime
		LL_LPUART_SetDEAssertionTime
	DEDT	LL_LPUART_GetDEDeassertionTime
		LL_LPUART_SetDEDeassertionTime
	FIFOEN	LL_LPUART_DisableFIFO
		LL_LPUART_EnableFIFO
		LL_LPUART_IsEnabledFIFO
	IDLEIE	LL_LPUART_DisableIT_IDLE
		LL_LPUART_EnableIT_IDLE
		LL_LPUART_IsEnabledIT_IDLE
	M	LL_LPUART_ConfigCharacter
		LL_LPUART_GetDataWidth
		LL_LPUART_SetDataWidth
	MME	LL_LPUART_DisableMuteMode
		LL_LPUART_EnableMuteMode
		LL_LPUART_IsEnabledMuteMode
	PCE	LL_LPUART_ConfigCharacter
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	PEIE	LL_LPUART_DisableIT_PE
		LL_LPUART_EnableIT_PE

Register	Field	Function
CR1	PEIE	LL_LPUART_IsEnabledIT_PE
	PS	LL_LPUART_ConfigCharacter
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	RE	LL_LPUART_DisableDirectionRx
		LL_LPUART_EnableDirectionRx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	RXFFIE	LL_LPUART_DisableIT_RXFF
		LL_LPUART_EnableIT_RXFF
		LL_LPUART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	LL_LPUART_DisableIT_RXNE_RXFNE
		LL_LPUART_EnableIT_RXNE_RXFNE
		LL_LPUART_IsEnabledIT_RXNE_RXFNE
	TCIE	LL_LPUART_DisableIT_TC
		LL_LPUART_EnableIT_TC
		LL_LPUART_IsEnabledIT_TC
	TE	LL_LPUART_DisableDirectionTx
		LL_LPUART_EnableDirectionTx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	TXEIE_TXFNFIE	LL_LPUART_DisableIT_TXE_TXFNF
		LL_LPUART_EnableIT_TXE_TXFNF
		LL_LPUART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_LPUART_DisableIT_TXFE
		LL_LPUART_EnableIT_TXFE
		LL_LPUART_IsEnabledIT_TXFE
	UE	LL_LPUART_Disable
		LL_LPUART_Enable
		LL_LPUART_IsEnabled
	UESM	LL_LPUART_DisableInStopMode
		LL_LPUART_EnableInStopMode
		LL_LPUART_IsEnabledInStopMode
	WAKE	LL_LPUART_GetWakeUpMethod
		LL_LPUART_SetWakeUpMethod
CR2	ADD	LL_LPUART_ConfigNodeAddress
		LL_LPUART_GetNodeAddress
	ADDM7	LL_LPUART_ConfigNodeAddress
		LL_LPUART_GetNodeAddressLen
	DATAINV	LL_LPUART_GetBinaryDataLogic
		LL_LPUART_SetBinaryDataLogic

Register	Field	Function
CR2	MSBFIRST	LL_LPUART_GetTransferBitOrder
		LL_LPUART_SetTransferBitOrder
	RXINV	LL_LPUART_GetRXPinLevel
		LL_LPUART_SetRXPinLevel
	STOP	LL_LPUART_ConfigCharacter
		LL_LPUART_GetStopBitsLength
		LL_LPUART_SetStopBitsLength
	SWAP	LL_LPUART_GetTXRXSwap
		LL_LPUART_SetTXRXSwap
	TXINV	LL_LPUART_GetTXPinLevel
		LL_LPUART_SetTXPinLevel
CR3	CTSE	LL_LPUART_DisableCTSHWFlowCtrl
		LL_LPUART_EnableCTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	CTSIE	LL_LPUART_DisableIT_CTS
		LL_LPUART_EnableIT_CTS
		LL_LPUART_IsEnabledIT_CTS
	DDRE	LL_LPUART_DisableDMADeactOnRxErr
		LL_LPUART_EnableDMADeactOnRxErr
		LL_LPUART_IsEnabledDMADeactOnRxErr
	DEM	LL_LPUART_DisableDEMode
		LL_LPUART_EnableDEMode
		LL_LPUART_IsEnabledDEMode
	DEP	LL_LPUART_GetDESignalPolarity
		LL_LPUART_SetDESignalPolarity
	DMAR	LL_LPUART_DisableDMAReq_RX
		LL_LPUART_EnableDMAReq_RX
		LL_LPUART_IsEnabledDMAReq_RX
	DMAT	LL_LPUART_DisableDMAReq_TX
		LL_LPUART_EnableDMAReq_TX
		LL_LPUART_IsEnabledDMAReq_TX
	EIE	LL_LPUART_DisableIT_ERROR
		LL_LPUART_EnableIT_ERROR
		LL_LPUART_IsEnabledIT_ERROR
	HDSEL	LL_LPUART_DisableHalfDuplex
		LL_LPUART_EnableHalfDuplex
		LL_LPUART_IsEnabledHalfDuplex
	OVRDIS	LL_LPUART_DisableOverrunDetect
		LL_LPUART_EnableOverrunDetect
		LL_LPUART_IsEnabledOverrunDetect

Register	Field	Function
CR3	RTSE	LL_LPUART_DisableRTSHWFlowCtrl
		LL_LPUART_EnableRTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	RXFTCFG	LL_LPUART_ConfigFIFOsThreshold
		LL_LPUART_GetRXFIFOThreshold
		LL_LPUART_SetRXFIFOThreshold
	RXFTIE	LL_LPUART_DisableIT_RXFT
		LL_LPUART_EnableIT_RXFT
		LL_LPUART_IsEnabledIT_RXFT
	TXFTCFG	LL_LPUART_ConfigFIFOsThreshold
		LL_LPUART_GetTXFIFOThreshold
		LL_LPUART_SetTXFIFOThreshold
	TXFTIE	LL_LPUART_DisableIT_TXFT
		LL_LPUART_EnableIT_TXFT
		LL_LPUART_IsEnabledIT_TXFT
ICR	CMCF	LL_LPUART_ClearFlag_CM
	CTSCF	LL_LPUART_ClearFlag_nCTS
	FECF	LL_LPUART_ClearFlag_FE
	IDLECF	LL_LPUART_ClearFlag_IDLE
	NECF	LL_LPUART_ClearFlag_NE
	ORECF	LL_LPUART_ClearFlag_ORE
	PECF	LL_LPUART_ClearFlag_PE
	TCCF	LL_LPUART_ClearFlag_TC
ISR	BUSY	LL_LPUART_IsActiveFlag_BUSY
	CMF	LL_LPUART_IsActiveFlag_CM
	CTS	LL_LPUART_IsActiveFlag_CTS
	CTSIF	LL_LPUART_IsActiveFlag_nCTS
	FE	LL_LPUART_IsActiveFlag_FE
	IDLE	LL_LPUART_IsActiveFlag_IDLE
	NE	LL_LPUART_IsActiveFlag_NE
	ORE	LL_LPUART_IsActiveFlag_ORE
	PE	LL_LPUART_IsActiveFlag_PE
	REACK	LL_LPUART_IsActiveFlag_REACK
	RWU	LL_LPUART_IsActiveFlag_RWU
	RXFF	LL_LPUART_IsActiveFlag_RXFF
	RXFT	LL_LPUART_IsActiveFlag_RXFT
	RXNE_RXFNE	LL_LPUART_IsActiveFlag_RXNE_RXFNE
	SBKF	LL_LPUART_IsActiveFlag_SBK
	TC	LL_LPUART_IsActiveFlag_TC
	TEACK	LL_LPUART_IsActiveFlag_TEACK

Register	Field	Function
ISR	TXE_TXFNF	LL_LPUART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_LPUART_IsActiveFlag_TXFE
	TXFT	LL_LPUART_IsActiveFlag_TXFT
PRESC	PRESCALER	LL_LPUART_GetPrescaler
		LL_LPUART_SetPrescaler
RDR	RDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_ReceiveData8
		LL_LPUART_ReceiveData9
RQR	MMRQ	LL_LPUART_RequestEnterMuteMode
	RXFRQ	LL_LPUART_RequestRxDataFlush
	SBKRQ	LL_LPUART_RequestBreakSending
	TXFRQ	LL_LPUART_RequestTxDataFlush
TDR	TDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_TransmitData8
		LL_LPUART_TransmitData9

124.21 OPAMP

Table 45. **Correspondence between OPAMP registers and OPAMP low-layer driver functions**

Register	Field	Function
CSR	CALON	LL_OPAMP_GetMode
		LL_OPAMP_SetMode
	CALOUT	LL_OPAMP_IsCalibrationOutputSet
	CALSEL	LL_OPAMP_GetCalibrationSelection
		LL_OPAMP_SetCalibrationSelection
	OPALPM	LL_OPAMP_SetPowerMode
	OPAMODE	LL_OPAMP_GetFunctionalMode
		LL_OPAMP_SetFunctionalMode
	OPAMPXEN	LL_OPAMP_Disable
		LL_OPAMP_Enable
		LL_OPAMP_IsEnabled
	OPARANGE	LL_OPAMP_GetCommonPowerRange
		LL_OPAMP_SetCommonPowerRange
	PGGAIN	LL_OPAMP_GetPGAGain
		LL_OPAMP_SetPGAGain
	USERTRIM	LL_OPAMP_GetTrimmingMode
		LL_OPAMP_SetTrimmingMode
	VMSEL	LL_OPAMP_GetInputInverting
		LL_OPAMP_SetInputInverting
	VPSEL	LL_OPAMP_GetInputNonInverting
		LL_OPAMP_SetInputNonInverting

Register	Field	Function
LPOTR	TRIMLPOFFSETN	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
	TRIMLPOFFSETP	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
OTR	TRIMOFFSETN	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue
	TRIMOFFSETP	LL_OPAMP_GetTrimmingValue
		LL_OPAMP_SetTrimmingValue

124.22 PKA

Table 46. **Correspondence between PKA registers and PKA low-layer driver functions**

Register	Field	Function
CLRFR	ADDRERRFC	LL_PKA_ClearFlag_ADDERR
	OPERRFC	LL_PKA_ClearFlag_OPERR
	PROCENDFC	LL_PKA_ClearFlag_PROCEND
	RAMERRFC	LL_PKA_ClearFlag_RAMERR
CR	ADDRERRIE	LL_PKA_DisableIT_ADDERR
		LL_PKA_EnableIT_ADDRERR
		LL_PKA_IsEnabledIT_ADDRERR
	EN	LL_PKA_Disable
		LL_PKA_Enable
		LL_PKA_IsEnabled
	MODE	LL_PKA_Config
		LL_PKA_GetMode
		LL_PKA_SetMode
	OPERRIE	LL_PKA_EnableIT_OPERR
		LL_PKA_IsEnabledIT_OPERR
	PROCENDIE	LL_PKA_DisableIT_PROCEND
		LL_PKA_EnableIT_PROCEND
		LL_PKA_IsEnabledIT_PROCEND
	RAMERRIE	LL_PKA_DisableIT_RAMERR
		LL_PKA_EnableIT_RAMERR
		LL_PKA_IsEnabledIT_RAMERR
	START	LL_PKA_Start
SR	ADDRERRF	LL_PKA_IsActiveFlag_ADDRERR
	BUSY	LL_PKA_IsActiveFlag_BUSY
	OPERRF	LL_PKA_IsActiveFlag_OPERR
	PROCENDF	LL_PKA_IsActiveFlag_PROCEND
	RAMERRF	LL_PKA_IsActiveFlag_RAMERR

124.23 PWR

Table 47. Correspondence between PWR registers and PWR low-layer driver functions

Register	Field	Function
APCR	APC	LL_PWR_DisablePUPDConfig
		LL_PWR_EnablePUPDConfig
		LL_PWR_IsEnabledPUPDConfig
BDCR1	BREN	LL_PWR_DisableBkUpRegulator
		LL_PWR_EnableBkUpRegulator
		LL_PWR_IsEnabledBkUpRegulator
	MONEN	LL_PWR_DisableMonitoring
		LL_PWR_EnableMonitoring
		LL_PWR_IsEnabledMonitoring
BDCR2	VBE	LL_PWR_DisableBatteryCharging
		LL_PWR_EnableBatteryCharging
		LL_PWR_IsEnabledBatteryCharging
	VBRS	LL_PWR_GetBattChargResistor
		LL_PWR_SetBattChargResistor
BDSR	TEMPH	LL_PWR_IsActiveFlag_TEMPH
	TEMPL	LL_PWR_IsActiveFlag_TEMPL
	VBATH	LL_PWR_IsActiveFlag_VBATH
CR1	FORCE_USBPWR	LL_PWR_DisableOTGHSPHYLowPowerRetention
		LL_PWR_EnableOTGHSPHYLowPowerRetention
		LL_PWR_IsEnabledOTGHSPHYLowPowerRetention
	LPMS	LL_PWR_GetPowerMode
		LL_PWR_SetPowerMode
	RRSB1	LL_PWR_GetSRAM2SBRetention
		LL_PWR_SetSRAM2SBRetention
	RRSB2	LL_PWR_GetSRAM2SBRetention
		LL_PWR_SetSRAM2SBRetention
	SRAM1PD	LL_PWR_GetSRAM1RunRetention
		LL_PWR_SetSRAM1RunRetention
	SRAM2PD	LL_PWR_GetSRAM2RunRetention
		LL_PWR_SetSRAM2RunRetention
	SRAM3PD	LL_PWR_GetSRAM3RunRetention
		LL_PWR_SetSRAM3RunRetention
	SRAM4PD	LL_PWR_GetSRAM4RunRetention
		LL_PWR_SetSRAM4RunRetention
	SRAM5PD	LL_PWR_GetSRAM5RunRetention
		LL_PWR_SetSRAM5RunRetention
	SRAM6PD	LL_PWR_GetSRAM6RunRetention
		LL_PWR_SetSRAM6RunRetention

Register	Field	Function
CR1	UPLMEN	LL_PWR_DisableUltraLowPowerMode
		LL_PWR_EnableUltraLowPowerMode
		LL_PWR_IsEnabledUltraLowPowerMode
CR2	DC1RAMPDS	LL_PWR_GetDCache1RAMStopRetention
		LL_PWR_SetDCache1RAMStopRetention
	DC2RAMPDS	LL_PWR_GetDCache2RAMStopRetention
		LL_PWR_SetDCache2RAMStopRetention
	DMA2DRAMPDS	LL_PWR_GetDMA2DRAMStopRetention
		LL_PWR_SetDMA2DRAMStopRetention
	DSIRAMPDS	LL_PWR_GetDSIRAMStopRetention
		LL_PWR_SetDSIRAMStopRetention
	FLASHFWU	LL_PWR_DisableFlashFastWakeUp
		LL_PWR_EnableFlashFastWakeUp
		LL_PWR_IsEnabledFlashFastWakeUp
	GPRAMPDS	LL_PWR_GetGraphicPeriphRAMStopRetention
		LL_PWR_SetGraphicPeriphRAMStopRetention
	ICRAMPDS	LL_PWR_GetICacheRAMStopRetention
		LL_PWR_SetICacheRAMStopRetention
	JPEGRAMPDS	LL_PWR_GetJPEGRAMStopRetention
		LL_PWR_SetJPEGRAMStopRetention
	PKARAMPDS	LL_PWR_GetPKARAMStopRetention
		LL_PWR_SetPKARAMStopRetention
	PRAMPDS	LL_PWR_GetPeriphRAMStopRetention
		LL_PWR_SetPeriphRAMStopRetention
	SRAM1PDS1	LL_PWR_GetSRAM1StopRetention_1_3
		LL_PWR_SetSRAM1StopRetention_1_3
	SRAM1PDS10	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS11	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS12	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS2	LL_PWR_GetSRAM1StopRetention_1_3
		LL_PWR_SetSRAM1StopRetention_1_3
	SRAM1PDS3	LL_PWR_GetSRAM1StopRetention_1_3
		LL_PWR_SetSRAM1StopRetention_1_3
	SRAM1PDS4	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS5	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS6	LL_PWR_GetSRAM1StopRetention_4_12

Register	Field	Function
CR2	SRAM1PDS6	LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS7	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS8	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM1PDS9	LL_PWR_GetSRAM1StopRetention_4_12
		LL_PWR_SetSRAM1StopRetention_4_12
	SRAM2PDS1	LL_PWR_GetSRAM2StopRetention
		LL_PWR_SetSRAM2StopRetention
	SRAM2PDS2	LL_PWR_GetSRAM2StopRetention
		LL_PWR_SetSRAM2StopRetention
	SRAM3PDS1	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS10	LL_PWR_SetSRAM3StopRetention_9_13
	SRAM3PDS11	LL_PWR_SetSRAM3StopRetention_9_13
	SRAM3PDS12	LL_PWR_SetSRAM3StopRetention_9_13
	SRAM3PDS13	LL_PWR_SetSRAM3StopRetention_9_13
	SRAM3PDS2	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS3	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS4	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS5	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS6	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS7	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS8	LL_PWR_GetSRAM3StopRetention_1_8
		LL_PWR_SetSRAM3StopRetention_1_8
	SRAM3PDS9	LL_PWR_SetSRAM3StopRetention_9_13
	SRAM4FWU	LL_PWR_DisableSRAM4FastWakeUp
		LL_PWR_EnableSRAM4FastWakeUp
		LL_PWR_IsEnabledSRAM4FastWakeUp
	SRAM4PDS	LL_PWR_GetSRAM4StopRetention
		LL_PWR_SetSRAM4StopRetention
	SRDRUN	LL_PWR_GetSmartRunDomainMode
		LL_PWR_SetSmartRunDomainMode
CR3	FSTEN	LL_PWR_DisableFastSoftStart
		LL_PWR_EnableFastSoftStart

Register	Field	Function
CR3	FSTEN	LL_PWR_IsEnabledFastSoftStart
	REGSEL	LL_PWR_GetRegulatorSupply
		LL_PWR_SetRegulatorSupply
CR4	SRAM5PDS1	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS10	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS11	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS12	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS13	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS2	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS3	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS4	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS5	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS6	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS7	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS8	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
	SRAM5PDS9	LL_PWR_GetSRAM5StopRetention
		LL_PWR_SetSRAM5StopRetention
CR5	SRAM6PDS1	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
	SRAM6PDS2	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
	SRAM6PDS3	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
	SRAM6PDS4	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
	SRAM6PDS5	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
	SRAM6PDS6	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention

Register	Field	Function
CR5	SRAM6PDS7	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
	SRAM6PDS8	LL_PWR_GetSRAM6StopRetention
		LL_PWR_SetSRAM6StopRetention
DBPR	DBP	LL_PWR_DisableBkUpAccess
		LL_PWR_EnableBkUpAccess
		LL_PWR_IsEnabledBkUpAccess
PDCRx	PDx	LL_PWR_DisableGPIOPullDown
		LL_PWR_EnableGPIOPullDown
		LL_PWR_IsEnabledGPIOPullUp
PRIVCFGR	NSPRIV	LL_PWR_DisableNSecurePrivilege
		LL_PWR_EnableNSecurePrivilege
		LL_PWR_IsEnabledNSecurePrivilege
	SPRIV	LL_PWR_DisableSecurePrivilege
		LL_PWR_EnableSecurePrivilege
		LL_PWR_IsEnabledSecurePrivilege
PUCRx	PUx	LL_PWR_DisableGPIOPullUp
		LL_PWR_EnableGPIOPullUp
		LL_PWR_IsEnabledGPIOPullUp
SECCFGR	APCSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	LPMSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	VBSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	VDMSEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP1SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP2SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP3SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP4SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP5SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP6SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
	WUP7SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure

Register	Field	Function
SECCFGR	WUP8SEC	LL_PWR_ConfigSecure
		LL_PWR_GetConfigSecure
SR	CSSF	LL_PWR_ClearFlag_SB
		LL_PWR_ClearFlag_STOP
	SBF	LL_PWR_IsActiveFlag_SB
	STOPF	LL_PWR_IsActiveFlag_STOP
SVMCR	ASV	LL_PWR_DisableVddA
		LL_PWR_EnableVddA
		LL_PWR_IsEnabledVddA
	AVM1EN	LL_PWR_DisableVddAMonitor1
		LL_PWR_EnableVddAMonitor1
		LL_PWR_IsEnabledVddAMonitor1
	AVM2EN	LL_PWR_DisableVddAMonitor2
		LL_PWR_EnableVddAMonitor2
		LL_PWR_IsEnabledVddAMonitor2
	IO2SV	LL_PWR_DisableVddIO2
		LL_PWR_EnableVddIO2
		LL_PWR_IsEnabledVddIO2
	IO2VMEN	LL_PWR_DisableVddIO2Monitor
		LL_PWR_EnableVddIO2Monitor
		LL_PWR_IsEnabledVddIO2Monitor
	PVDE	LL_PWR_DisablePVD
		LL_PWR_EnablePVD
		LL_PWR_IsEnabledPVD
	PVDLS	LL_PWR_GetPVDLevel
		LL_PWR_SetPVDLevel
	USV	Section 113.1.1: LL_PWR_DisableVDDUSB
		LL_PWR_EnableVddUSB
		LL_PWR_IsEnabledVddUSB
	UVMEN	LL_PWR_DisableVddUSBMonitor
		LL_PWR_EnableVddUSBMonitor
		LL_PWR_IsEnabledVddUSBMonitor
SVMSR	ACTVOSRDY	LL_PWR_IsActiveFlag_ACTVOS
	ACTVOS[1:0]	LL_PWR_GetRegulCurrentVOS
	PVDO	LL_PWR_IsActiveFlag_PVDO
	REGS	LL_PWR_IsActiveFlag_REGULATOR
	VDDA1RDY	LL_PWR_IsActiveFlag_VDDA1
	VDDA2RDY	LL_PWR_IsActiveFlag_VDDA2
	VDDIO2RDY	LL_PWR_IsActiveFlag_VDDIO2
	VDDUSBRDY	LL_PWR_IsActiveFlag_VDDUSB
UCPDR	UCPD_DBDIS	LL_PWR_DisableUCPDDeadBattery

Register	Field	Function
UCPDR	UCPD_DBDIS	LL_PWR_EnableUCPDDeadBattery
		LL_PWR_IsEnabledUCPDDeadBattery
	UCPD_STDBY	LL_PWR_DisableUCPDStandbyMode
		LL_PWR_EnableUCPDStandbyMode
		LL_PWR_IsEnabledUCPDStandbyMode
VOSR	BOOSTEN	LL_PWR_DisableEPODBooster
		LL_PWR_EnableEPODBooster
		LL_PWR_IsEnabledEPODBooster
	BOOSTRDY	LL_PWR_IsActiveFlag_BOOST
	USBBOOSTEN	LL_PWR_DisableUSBEPODBooster
		LL_PWR_EnableUSBEPODBooster
		LL_PWR_IsEnabledUSBEPODBooster
	USBBOOSTRDY	LL_PWR_IsActiveFlag_USBBOOST
	USBPWREN	LL_PWR_DisableUSBPowerSupply
		LL_PWR_EnableUSBPowerSupply
		LL_PWR_IsEnabledUSBPowerSupply
	VDD11USBDIS	LL_PWR_DisableVDD11USB
		LL_PWR_EnableVDD11USB
		LL_PWR_IsEnabledVDD11USB
	VOS	LL_PWR_GetRegulVoltageScaling
		LL_PWR_SetRegulVoltageScaling
	VOSRDY	LL_PWR_IsActiveFlag_VOS
WUCR1	WUPENx	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
WUCR2	WUPPx	LL_PWR_GetWakeUpPinPolarity
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
WUCR3	WUSELx	LL_PWR_GetWakeUpPinSignalSelection
		LL_PWR_SetWakeUpPinSignal0Selection
		LL_PWR_SetWakeUpPinSignal1Selection
		LL_PWR_SetWakeUpPinSignal2Selection
		LL_PWR_SetWakeUpPinSignal3Selection
WUSCR	CWUF	LL_PWR_ClearFlag_WU
	CWUF1	LL_PWR_ClearFlag_WU1
	CWUF2	LL_PWR_ClearFlag_WU2
	CWUF3	LL_PWR_ClearFlag_WU3
	CWUF4	LL_PWR_ClearFlag_WU4
	CWUF5	LL_PWR_ClearFlag_WU5
	CWUF6	LL_PWR_ClearFlag_WU6
	CWUF7	LL_PWR_ClearFlag_WU7

Register	Field	Function
WUSCR	CWUF8	LL_PWR_ClearFlag_WU8
WUSR	WUF1	LL_PWR_IsActiveFlag_WU1
	WUF2	LL_PWR_IsActiveFlag_WU2
	WUF3	LL_PWR_IsActiveFlag_WU3
	WUF4	LL_PWR_IsActiveFlag_WU4
	WUF5	LL_PWR_IsActiveFlag_WU5
	WUF6	LL_PWR_IsActiveFlag_WU6
	WUF7	LL_PWR_IsActiveFlag_WU7
	WUF8	LL_PWR_IsActiveFlag_WU8

124.24 RCC

Table 48. Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
BDCR	BDRST	LL_RCC_ForceBackupDomainReset
		LL_RCC_ReleaseBackupDomainReset
	LSCOEN	LL_RCC_LSCO_Disable
		LL_RCC_LSCO_Enable
	LSCOSEL	LL_RCC_LSCO_GetSource
		LL_RCC_LSCO_SetSource
	LSEBYP	LL_RCC_LSE_DisableBypass
		LL_RCC_LSE_EnableBypass
	LSECSSD	LL_RCC_LSE_IsCSSDetected
	LSECSSON	LL_RCC_LSE_DisableCSS
		LL_RCC_LSE_EnableCSS
	LSEDRV	LL_RCC_LSE_GetDriveCapability
		LL_RCC_LSE_SetDriveCapability
	LSEGFON	LL_RCC_LSE_DisableGlitchFilter
		LL_RCC_LSE_EnableGlitchFilter
		LL_RCC_LSE_IsGlitchFilterEnabled
	LSEON	LL_RCC_LSE_Disable
		LL_RCC_LSE_Enable
	LSERDY	LL_RCC_LSE_IsReady
	LSESYSEN	LL_RCC_LSE_DisablePropagation
		LL_RCC_LSE_EnablePropagation
	LSESYSRDY	LL_RCC_LSESYS_IsReady
		LL_RCC_LSE_IsPropagationReady
	LSION	LL_RCC_LSI_Disable
		LL_RCC_LSI_Enable
	LSIPREDIV	LL_RCC_LSI_GetPrescaler
		LL_RCC_LSI_SetPrescaler

Register	Field	Function
BDCR	LSIRDY	LL_RCC_LSI_IsReady
	RTCEN	LL_RCC_DisableRTC
		LL_RCC_EnableRTC
		LL_RCC_IsEnabledRTC
	RTCSEL	LL_RCC_GetRTCClockSource
		LL_RCC_SetRTCClockSource
CCIPR1	FDCANSEL	LL_RCC_GetFDCANClockSource
		LL_RCC_SetFDCANClockSource
	I2C1SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	I2C2SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	I2C4SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	ICLKSEL	LL_RCC_GetSDMMCClockSource
		LL_RCC_GetUSBClockSource
		LL_RCC_SetSDMMCClockSource
		LL_RCC_SetUSBClockSource
	LPTIM2SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPUART1SEL	LL_RCC_GetLPUARTClockSource
	SPI1SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SPI2SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
	SYSTICKSEL	LL_RCC_SetSystickClockSource
	TIMICSEL	LL_RCC_GetTIMICClockSource
		LL_RCC_SetTIMICClockSource
	UART4SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	UART5SEL	LL_RCC_GetUARTClockSource
		LL_RCC_SetUARTClockSource
	USART1SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART2SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USART3SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
CCIPR2	DSISEL	LL_RCC_SetDSIClockSource
	HSPISEL	LL_RCC_GetHSPIClockSource
		LL_RCC_SetHSPIClockSource

Register	Field	Function
CCIPR2	LTDCSEL	LL_RCC_SetLTDCClockSource
	MDF1SEL	LL_RCC_GetMDF1ClockSource
	OSPISEL	LL_RCC_GetOCTOSPIClockSource
		LL_RCC_SetOCTOSPIClockSource
	RNGSEL	LL_RCC_GetRNGClockSource
		LL_RCC_SetRNGClockSource
	SAESSEL	LL_RCC_SetSAESClockSource
	SAI1SEL	LL_RCC_GetSAIClockSource
		LL_RCC_SetSAIClockSource
	SAI2SEL	LL_RCC_GetSAIClockSource
		LL_RCC_SetSAIClockSource
	SDMMCSEL	LL_RCC_GetSDMMCKernelClockSource
		LL_RCC_SetSDMMCKernelClockSource
	USART6SEL	LL_RCC_GetUSARTClockSource
		LL_RCC_SetUSARTClockSource
	USBPHYCSEL	LL_RCC_GetUSBPHYClockSource
		LL_RCC_SetUSBPHYClockSource
CCIPR3	ADCDACSEL	LL_RCC_SetADCDACClockSource
	ADF1SEL	LL_RCC_GetADF1ClockSource
		LL_RCC_SetADF1ClockSource
	DAC1SEL	LL_RCC_GetDAC1ClockSource
		LL_RCC_SetDAC1ClockSource
	I2C3SEL	LL_RCC_GetI2CClockSource
		LL_RCC_SetI2CClockSource
	LPTIM2SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPTIM34SEL	LL_RCC_GetLPTIMClockSource
		LL_RCC_SetLPTIMClockSource
	LPUART1SEL	LL_RCC_SetLPUARTClockSource
	MDF1SEL	LL_RCC_SetMDF1ClockSource
	SPI3SEL	LL_RCC_GetSPIClockSource
		LL_RCC_SetSPIClockSource
CFGR1	MCOPRE	LL_RCC_ConfigMCO
	MCOSEL	LL_RCC_ConfigMCO
	STOPKERWUCK	LL_RCC_GetKerClkAfterWakeFromStop
		LL_RCC_SetKerClkAfterWakeFromStop
	STOPWUCK	LL_RCC_GetClkAfterWakeFromStop
		LL_RCC_SetClkAfterWakeFromStop
	SW	LL_RCC_SetSysClkSource
	SWS	LL_RCC_GetSysClkSource
CFGR2	HPRE	LL_RCC_GetAHBPrescaler

Register	Field	Function
CFGR2	HPRE	LL_RCC_SetAHBPrescaler
	PPRE1	LL_RCC_GetAPB1Prescaler
		LL_RCC_SetAPB1Prescaler
	PPRE2	LL_RCC_GetAPB2Prescaler
		LL_RCC_SetAPB2Prescaler
	PPRE_DPHY	LL_RCC_GetDPHYPrescaler
		LL_RCC_SetDPHYPrescaler
CFGR3	PPRE3	LL_RCC_GetAPB2Prescaler
		LL_RCC_SetAPB3Prescaler
CICR	CSSC	LL_RCC_ClearFlag_HSECSS
	HSERDYC	LL_RCC_ClearFlag_HSERDY
	HSI48RDYC	LL_RCC_ClearFlag_HSI48RDY
	HSIRDYC	LL_RCC_ClearFlag_HSIRDY
	LSERDYC	LL_RCC_ClearFlag_LSERDY
	LSIRDYC	LL_RCC_ClearFlag_LSIRDY
	MSIKRDYC	LL_RCC_ClearFlag_MSIKRDY
	MSISRDYC	LL_RCC_ClearFlag_MSIRDY
	PLL1RDYC	LL_RCC_ClearFlag_PLL1RDY
	PLL2RDYC	LL_RCC_ClearFlag_PLL2RDY
	PLL3RDYC	LL_RCC_ClearFlag_PLL3RDY
	SHSIRDYC	LL_RCC_ClearFlag_SHSIRDY
CIER	HSERDYIE	LL_RCC_DisableIT_HSERDY
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY
	HSI48RDYIE	LL_RCC_DisableIT_HSI48RDY
		LL_RCC_EnableIT_HSI48RDY
		LL_RCC_IsEnabledIT_HSI48RDY
	HSIRDYIE	LL_RCC_DisableIT_HSIRDY
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSERDYIE	LL_RCC_DisableIT_LSERDY
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYIE	LL_RCC_DisableIT_LSIRDY
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
	MSIKRDYIE	LL_RCC_DisableIT_MSIKRDY
		LL_RCC_EnableIT_MSIKRDY
		LL_RCC_IsEnabledIT_MSIKRDY
	MSIRDYIE	LL_RCC_EnableIT_MSIRDY
		LL_RCC_IsEnabledIT_MSIRDY

Register	Field	Function
CIER	MSISRDYIE	LL_RCC_DisableIT_MSIRDY
	PLL1RDYIE	LL_RCC_DisableIT_PLL1RDY
		LL_RCC_EnableIT_PLL1RDY
		LL_RCC_IsEnabledIT_PLL1RDY
	PLL2RDYIE	LL_RCC_DisableIT_PLL2RDY
		LL_RCC_EnableIT_PLL2RDY
		LL_RCC_IsEnabledIT_PLL2RDY
	PLL3RDYIE	LL_RCC_DisableIT_PLL3RDY
		LL_RCC_EnableIT_PLL3RDY
		LL_RCC_IsEnabledIT_PLL3RDY
	SHSIRDYIE	LL_RCC_DisableIT_SHSIRDY
		LL_RCC_EnableIT_SHSIRDY
		LL_RCC_IsEnabledIT_SHSIRDY
CIFR	CSSF	LL_RCC_IsActiveFlag_HSECSS
	HSERDYF	LL_RCC_IsActiveFlag_HSERDY
	HSI48RDYF	LL_RCC_IsActiveFlag_HSI48RDY
	HSIRDYF	LL_RCC_IsActiveFlag_HSIRDY
	LSERDYF	LL_RCC_IsActiveFlag_LSERDY
	LSIRDYF	LL_RCC_IsActiveFlag_LSIRDY
	MSIKRDYF	LL_RCC_IsActiveFlag_MSIKRDY
	MSISRDYF	LL_RCC_IsActiveFlag_MSIRDY
	PLL1RDYF	LL_RCC_IsActiveFlag_PLL1RDY
	PLL2RDYF	LL_RCC_IsActiveFlag_PLL2RDY
	PLL3RDYF	LL_RCC_IsActiveFlag_PLL3RDY
	SHSIRDYF	LL_RCC_IsActiveFlag_SHSIRDY
CR	CSSON	LL_RCC_HSE_EnableCSS
	HSEBYP	LL_RCC_HSE_DisableBypass
		LL_RCC_HSE_EnableBypass
	HSEEXT	LL_RCC_HSE_GetClockMode
		LL_RCC_HSE_SetClockMode
	HSEON	LL_RCC_HSE_Disable
		LL_RCC_HSE_Enable
	HSERDY	LL_RCC_HSE_IsReady
	HSI48ON	LL_RCC_HSI48_Disable
		LL_RCC_HSI48_Enable
	HSI48RDY	LL_RCC_HSI48_IsReady
	HSIKERON	LL_RCC_HSI_DisableInStopMode
		LL_RCC_HSI_EnableInStopMode
		LL_RCC_HSI_IsEnabledInStopMode
	HSION	LL_RCC_HSI_Disable
		LL_RCC_HSI_Enable

Register	Field	Function
CR	HSIRDY	LL_RCC_HSI_IsReady
	MSIKERON	LL_RCC_MSIK_DisableInStopMode
		LL_RCC_MSIK_EnableInStopMode
		LL_RCC_MSIK_IsEnabledInStopMode
	MSIKON	LL_RCC_MSIK_Disable
		LL_RCC_MSIK_Enable
	MSIKRDY	LL_RCC_MSIK_IsReady
	MSIPLLEN	LL_RCC_IsEnabledPLLMode
		LL_RCC_MSI_DisablePLLMode
		LL_RCC_MSI_EnablePLLMode
	MSIPLLFAST	LL_RCC_Disable_MSIPLLFAST
		LL_RCC_Enable_MSIPLLFAST
		LL_RCC_MSI_IsEnabledMSIPLLFAST
	MSIPLLSEL	LL_RCC_GetMSIPLLMode
		LL_RCC_SetMSIPLLMode
	MSISON	LL_RCC_MSIS_Disable
		LL_RCC_MSIS_Enable
	MSISRDY	LL_RCC_MSIS_IsReady
	PLL1ON	LL_RCC_PLL1_Enable
	PLL1RDY	LL_RCC_PLL1_IsReady
	PLL2ON	LL_RCC_PLL2_Disable
		LL_RCC_PLL2_Enable
	PLL2RDY	LL_RCC_PLL2_IsReady
	PLL3ON	LL_RCC_PLL3_Disable
		LL_RCC_PLL3_Enable
	PLL3RDY	LL_RCC_PLL3_IsReady
	PLLON	LL_RCC_PLL1_Disable
	PRIV	LL_RCC_DisablePrivilegedMode
		LL_RCC_IsEnabledPrivilegedMode
	SHSION	LL_RCC_SHSI_Disable
		LL_RCC_SHSI_Enable
	SHSIRDY	LL_RCC_SHSI_IsReady
CRRCR	HSI48CAL	LL_RCC_HSI48_GetCalibration
CSR	BORRSTF	LL_RCC_IsActiveFlag_BORRST
	IWDGRSTF	LL_RCC_IsActiveFlag_IWDGRST
	LPWRRSTF	LL_RCC_IsActiveFlag_LPWRRST
	MSIKSRANGE	LL_RCC_MSIK_GetRangeAfterStandby
		LL_RCC_MSIK_SetRangeAfterStandby
	MSISSRANGE	LL_RCC_MSIS_GetRangeAfterStandby
		LL_RCC_MSIS_SetRangeAfterStandby
	OBLRSTF	LL_RCC_IsActiveFlag_OBLRST

Register	Field	Function
CSR	PINRSTF	LL_RCC_IsActiveFlag_PINRST
	RMVF	LL_RCC_ClearResetFlags
	SFTRSTF	LL_RCC_IsActiveFlag_SFTRST
	WWDGRSTF	LL_RCC_IsActiveFlag_WWDGRST
ICSC3R	HSITRIM	LL_RCC_HSI_GetCalibTrimming
ICSCR1	MSIBIAS	LL_RCC_MSI_GetMSIBiasMode
		LL_RCC_MSI_SetMSIBiasMode
	MSICALx	LL_RCC_MSI_GetCalibration
	MSIKRANGE	LL_RCC_MSIK_GetRange
		LL_RCC_MSIK_SetRange
	MSIRGSEL	LL_RCC_MSI_EnableRangeSelection
		LL_RCC_MSI_IsEnabledRangeSelect
	MSISRANGE	LL_RCC_MSIS_GetRange
		LL_RCC_MSIS_SetRange
ICSCR2	MSITRIMx	LL_RCC_MSI_GetCalibTrimming
		LL_RCC_MSI_SetCalibTrimming
ICSCR3	HSICAL	LL_RCC_HSI_GetCalibration
	HSITRIM	LL_RCC_HSI_SetCalibTrimming
PLL1CFGR	PLL1FRACEN	LL_RCC_PLL1FRACN_Disable
		LL_RCC_PLL1FRACN_Enable
		LL_RCC_PLL1FRACN_IsEnabled
	PLL1M	LL_RCC_PLL1_ConfigDomain_48M
		LL_RCC_PLL1_ConfigDomain_SYS
		LL_RCC_PLL1_GetDivider
		LL_RCC_PLL1_SetDivider
	PLL1MBOOST	LL_RCC_GetPll1EPodPrescaler
		LL_RCC_SetPll1EPodPrescaler
	PLL1N	LL_RCC_PLL1_ConfigDomain_48M
		LL_RCC_PLL1_ConfigDomain_SYS
		LL_RCC_PLL1_GetN
		LL_RCC_PLL1_SetN
	PLL1P	LL_RCC_PLL1_GetP
		LL_RCC_PLL1_SetP
	PLL1PEN	LL_RCC_PLL1_DisableDomain_SAI
		LL_RCC_PLL1_EnableDomain_SAI
		LL_RCC_PLL1_IsEnabledDomain_SAI
	PLL1Q	LL_RCC_PLL1_ConfigDomain_48M
		LL_RCC_PLL1_GetQ
	PLL1QEN	LL_RCC_PLL1_DisableDomain_48M
		LL_RCC_PLL1_EnableDomain_48M
		LL_RCC_PLL1_IsEnabledDomain_48M

Register	Field	Function
PLL1CFGR	PLL1R	LL_RCC_PLL1_ConfigDomain_SYS
	PLL1REN	LL_RCC_PLL1_DisableDomain_SYS
		LL_RCC_PLL1_EnableDomain_SYS
		LL_RCC_PLL1_IsEnabledDomain_SYS
	PLL1RGE	LL_RCC_PLL1_SetVCOInputRange
	PLL1SRC	LL_RCC_PLL1_ConfigDomain_48M
		LL_RCC_PLL1_ConfigDomain_SYS
		LL_RCC_PLL1_GetMainSource
		LL_RCC_PLL1_SetMainSource
PLL1DIVR	PLL1R	LL_RCC_PLL1_GetR
		LL_RCC_PLL1_SetR
PLL1FRACR	PLL1FRACN	LL_RCC_PLL1_GetFRACN
		LL_RCC_PLL1_SetFRACN
PLL2CFGR	PLL2FRACEN	LL_RCC_PLL2FRACN_Disable
		LL_RCC_PLL2FRACN_Enable
		LL_RCC_PLL2FRACN_IsEnabled
	PLL2M	LL_RCC_PLL2_ConfigDomain_48M
		LL_RCC_PLL2_ConfigDomain_ADC
		LL_RCC_PLL2_GetDivider
		LL_RCC_PLL2_SetDivider
	PLL2N	LL_RCC_PLL2_ConfigDomain_48M
		LL_RCC_PLL2_ConfigDomain_ADC
		LL_RCC_PLL2_GetN
		LL_RCC_PLL2_SetN
	PLL2P	LL_RCC_PLL2_GetP
		LL_RCC_PLL2_SetP
	PLL2PEN	LL_RCC_PLL2_DisableDomain_SAI
		LL_RCC_PLL2_EnableDomain_SAI
		LL_RCC_PLL2_IsEnabledDomain_SAI
	PLL2Q	LL_RCC_PLL2_ConfigDomain_48M
		LL_RCC_PLL2_GetQ
	PLL2QEN	LL_RCC_PLL2_DisableDomain_48M
		LL_RCC_PLL2_EnableDomain_48M
		LL_RCC_PLL2_IsEnabledDomain_48M
	PLL2R	LL_RCC_PLL2_ConfigDomain_ADC
		LL_RCC_PLL2_SetR
	PLL2REN	LL_RCC_PLL2_DisableDomain_ADC
		LL_RCC_PLL2_EnableDomain_ADC
		LL_RCC_PLL2_IsEnabledDomain_ADC
	PLL2RGE	LL_RCC_PLL2_SetVCOInputRange
	PLL2SRC	LL_RCC_PLL2_ConfigDomain_48M

Register	Field	Function
PLL2CFGR	PLL2SRC	LL_RCC_PLL2_ConfigDomain_ADC
		LL_RCC_PLL2_GetSource
		LL_RCC_PLL2_SetSource
PLL2DIVR	PLL2R	LL_RCC_PLL2_GetR
PLL2FRACR	PLL2FRACN	LL_RCC_PLL2_GetFRACN
		LL_RCC_PLL2_SetFRACN
PLL3CFGR	PLL3FRACEN	LL_RCC_PLL3FRACN_Disable
		LL_RCC_PLL3FRACN_Enable
		LL_RCC_PLL3FRACN_IsEnabled
	PLL3M	LL_RCC_PLL3_ConfigDomain_48M
		LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
		LL_RCC_PLL3_ConfigDomain_SAI
		LL_RCC_PLL3_GetDivider
		LL_RCC_PLL3_SetDivider
	PLL3N	LL_RCC_PLL3_ConfigDomain_48M
		LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
		LL_RCC_PLL3_ConfigDomain_SAI
		LL_RCC_PLL3_GetN
		LL_RCC_PLL3_SetN
	PLL3P	LL_RCC_PLL3_GetP
		LL_RCC_PLL3_SetP
	PLL3PEN	LL_RCC_PLL3_DisableDomain_SAI
		LL_RCC_PLL3_EnableDomain_SAI
		LL_RCC_PLL3_IsEnabledDomain_SAI
	PLL3Q	LL_RCC_PLL3_ConfigDomain_48M
		LL_RCC_PLL3_GetQ
	PLL3QEN	LL_RCC_PLL3_DisableDomain_48M
		LL_RCC_PLL3_EnableDomain_48M
		LL_RCC_PLL3_IsEnabledDomain_48M
	PLL3R	LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
		LL_RCC_PLL3_SetR
	PLL3REN	LL_RCC_PLL3_DisableDomain_HSPI_LTDC
		LL_RCC_PLL3_EnableDomain_HSPI_LTDC
		LL_RCC_PLL3_IsEnabledDomain_HSPI_LTDC
	PLL3RGE	LL_RCC_PLL3_SetVCOInputRange
	PLL3SRC	LL_RCC_PLL3_ConfigDomain_48M
		LL_RCC_PLL3_ConfigDomain_HSPI_LTDC
		LL_RCC_PLL3_ConfigDomain_SAI
		LL_RCC_PLL3_GetSource
		LL_RCC_PLL3_SetSource
PLL3DIVR	PLL3P	LL_RCC_PLL3_ConfigDomain_SAI

Register	Field	Function
PLL3DIVR	PLL3R	LL_RCC_PLL3_GetR
PLL3FRACR	PLL3FRACN	LL_RCC_PLL3_GetFRACN
		LL_RCC_PLL3_SetFRACN
PLLC1FGR	PLL1M	LL_RCC_PLL1_ConfigDomain_SAI
	PLL1N	LL_RCC_PLL1_ConfigDomain_SAI
	PLL1P	LL_RCC_PLL1_ConfigDomain_SAI
	PLL1SRC	LL_RCC_PLL1_ConfigDomain_SAI
PLLC2FGR	PLL2M	LL_RCC_PLL2_ConfigDomain_SAI
	PLL2N	LL_RCC_PLL2_ConfigDomain_SAI
	PLL2P	LL_RCC_PLL2_ConfigDomain_SAI
	PLL2SRC	LL_RCC_PLL2_ConfigDomain_SAI
PLLCFGR	PLL1Q	LL_RCC_PLL1_SetQ
	PLL2Q	LL_RCC_PLL2_SetQ
	PLL3Q	LL_RCC_PLL3_SetQ
PRIVCFGR	NSPRIV	LL_RCC_DisableNSecPrivilegedMode
		LL_RCC_EnableNSecPrivilegedMode
		LL_RCC_EnablePrivilegedMode
		LL_RCC_IsEnabledNSecPrivilegedMode
	SPRIV	LL_RCC_DisableSecPrivilegedMode
		LL_RCC_EnablePrivilegedMode
		LL_RCC_EnableSecPrivilegedMode
		LL_RCC_IsEnabledSecPrivilegedMode
SECCFGR	CLK48MSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	HSESEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	HSI48SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	HSISEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	LSESEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	LSISEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	MSISEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PLL1SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PLL2SEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	PLL3SEC	LL_RCC_ConfigSecure

Register	Field	Function
SECCFGR	PLL3SEC	LL_RCC_GetConfigSecure
	PRESCSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	RMVFSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure
	SYSCLKSEC	LL_RCC_ConfigSecure
		LL_RCC_GetConfigSecure

124.25 RNG

Table 49. Correspondence between RNG registers and RNG low-layer driver functions

Register	Field	Function
CR	CED	LL_RNG_DisableClkErrorDetect
		LL_RNG_EnableClkErrorDetect
		LL_RNG_IsEnabledClkErrorDetect
	CLKDIV	LL_RNG_GetClockDivider
		LL_RNG_SetClockDivider
	CONDRST	LL_RNG_DisableCondReset
		LL_RNG_EnableCondReset
		LL_RNG_IsEnabledCondReset
	CONFIGLOCK	LL_RNG_ConfigLock
		LL_RNG_IsConfigLocked
	IE	LL_RNG_DisableIT
		LL_RNG_EnableIT
		LL_RNG_IsEnabledIT
	NISTC	LL_RNG_DisableNistCompliance
		LL_RNG_EnableNistCompliance
		LL_RNG_IsEnabledNistCompliance
	RNGEN	LL_RNG_Disable
		LL_RNG_Enable
		LL_RNG_IsEnabled
	RNG_CONFIG1	LL_RNG_GetConfig1
		LL_RNG_SetConfig1
	RNG_CONFIG2	LL_RNG_GetConfig2
		LL_RNG_SetConfig2
	RNG_CONFIG3	LL_RNG_GetConfig3
		LL_RNG_SetConfig3
DR	RNDATA	LL_RNG_ReadRandData32
HTCR	HTCFG	LL_RNG_GetHealthConfig
		LL_RNG_SetHealthConfig
SR	CECS	LL_RNG_IsActiveFlag_CECS

Register	Field	Function
SR	CEIS	LL_RNG_ClearFlag_CEIS
		LL_RNG_IsActiveFlag_CEIS
	DRDY	LL_RNG_IsActiveFlag_DRDY
	SECS	LL_RNG_IsActiveFlag_SECS
	SEIS	LL_RNG_ClearFlag_SEIS
		LL_RNG_IsActiveFlag_SEIS

124.26 RTC

Table 50. **Correspondence between RTC registers and RTC low-layer driver functions**

Register	Field	Function
RCT_ALRMASSR	SS	LL_RTC_ALMA_GetSubSecond
		LL_RTC_ALMA_SetSubSecond
RTC_ALRABINR	SSCLR	LL_RTC_ALMA_GetBinAutoClr
		LL_RTC_ALMA_SetBinAutoClr
RTC_ALRBBINR	SSCLR	LL_RTC_ALMB_GetBinAutoClr
		LL_RTC_ALMB_SetBinAutoClr
RTC_ALRMAR	DT	LL_RTC_ALMA_GetDay
		LL_RTC_ALMA_SetDay
	DU	LL_RTC_ALMA_GetDay
		LL_RTC_ALMA_GetWeekDay
		LL_RTC_ALMA_SetDay
		LL_RTC_ALMA_SetWeekDay
	HT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	HU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	MNT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MNU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MSK1	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask

Register	Field	Function
RTC_ALRMAR	MSK2	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK3	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK4	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	PM	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetTimeFormat
		LL_RTC_ALMA_SetTimeFormat
	ST	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	SU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	WDSEL	LL_RTC_ALMA_DisableWeekday
		LL_RTC_ALMA_EnableWeekday
RTC_ALRMASSR	MASKSS	LL_RTC_ALMA_GetSubSecondMask
		LL_RTC_ALMA_SetSubSecondMask
RTC_ALRMBR	DT	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_SetDay
	DU	LL_RTC_ALMB_GetDay
		LL_RTC_ALMB_GetWeekDay
		LL_RTC_ALMB_SetDay
		LL_RTC_ALMB_SetWeekDay
	HT	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	HU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetHour
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetHour
	MNT	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetMinute
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MNU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetMinute

Register	Field	Function
RTC_ALRMBR	MNU	LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetMinute
	MSK1	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK2	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK3	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	MSK4	LL_RTC_ALMB_GetMask
		LL_RTC_ALMB_SetMask
	PM	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetTimeFormat
		LL_RTC_ALMB_SetTimeFormat
	ST	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	SU	LL_RTC_ALMB_ConfigTime
		LL_RTC_ALMB_GetSecond
		LL_RTC_ALMB_GetTime
		LL_RTC_ALMB_SetSecond
	WDSEL	LL_RTC_ALMB_DisableWeekday
		LL_RTC_ALMB_EnableWeekday
RTC_ALRMBSSR	MASKSS	LL_RTC_ALMB_GetSubSecondMask
		LL_RTC_ALMB_SetSubSecondMask
	SS	LL_RTC_ALMB_GetSubSecond
		LL_RTC_ALMB_SetSubSecond
RTC_CALR	CALM	LL_RTC_CAL_GetMinus
		LL_RTC_CAL_SetMinus
	CALP	LL_RTC_CAL_IsPulseInserted
		LL_RTC_CAL_SetPulse
	CALW16	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	CALW8	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	LPCAL	LL_RTC_CAL_LowPower_Disable
		LL_RTC_CAL_LowPower_Enable
		LL_RTC_CAL_LowPower_IsEnabled
RTC_CR	ADD1H	LL_RTC_TIME_IncHour
	ALRAE	LL_RTC_ALMA_Disable
		LL_RTC_ALMA_Enable

Register	Field	Function
RTC_CR	ALRAIE	LL_RTC_DisableIT_ALRA
		LL_RTC_EnableIT_ALRA
		LL_RTC_IsEnabledIT_ALRA
	ALRBE	LL_RTC_ALMB_Disable
		LL_RTC_ALMB_Enable
	ALRBIE	LL_RTC_DisableIT_ALRB
		LL_RTC_EnableIT_ALRB
		LL_RTC_IsEnabledIT_ALRB
	BKP	LL_RTC_TIME_DisableDayLightStore
		LL_RTC_TIME_EnableDayLightStore
		LL_RTC_TIME_IsDayLightStoreEnabled
	BYPSHAD	LL_RTC_DisableShadowRegBypass
		LL_RTC_EnableShadowRegBypass
		LL_RTC_IsShadowRegBypassEnabled
	COE	LL_RTC_CAL_GetOutputFreq
		LL_RTC_CAL_SetOutputFreq
	COSEL	LL_RTC_CAL_GetOutputFreq
		LL_RTC_CAL_SetOutputFreq
	FMT	LL_RTC_GetHourFormat
		LL_RTC_SetHourFormat
	ITSE	LL_RTC_TS_DisableInternalEvent
		LL_RTC_TS_EnableInternalEvent
	OSEL	LL_RTC_GetAlarmOutEvent
		LL_RTC_SetAlarmOutEvent
	OUT2EN	LL_RTC_DisableOutput2
		LL_RTC_EnableOutput2
		LL_RTC_IsOutput2Enabled
	POL	LL_RTC_GetOutputPolarity
		LL_RTC_SetOutputPolarity
	REFCKON	LL_RTC_DisableRefClock
		LL_RTC_EnableRefClock
	SSRUIE	LL_RTC_DisableIT_SSRU
		LL_RTC_EnableIT_SSRU
		LL_RTC_IsEnabledIT_SSRU
	SUB1H	LL_RTC_TIME_DecHour
	TAMPALRM_PU	LL_RTC_EnableAlarmPullUp
		LL_RTC_IsAlarmPullUpEnabled
	TAMPALRM_TYPE	LL_RTC_SetAlarmOutputType
	TAMPOE	LL_RTC_DisableTamperOutput
		LL_RTC_EnableTamperOutput
		LL_RTC_IsTamperOutputEnabled

Register	Field	Function
RTC_CR	TAMPTS	LL_RTC_TS_DisableOnTamper
		LL_RTC_TS_EnableOnTamper
	TSE	LL_RTC_TS_Disable
		LL_RTC_TS_Enable
	TSEDGE	LL_RTC_TS_GetActiveEdge
		LL_RTC_TS_SetActiveEdge
	TSIE	LL_RTC_DisableIT_TS
		LL_RTC_EnableIT_TS
		LL_RTC_IsEnabledIT_TS
	WUCKSEL	LL_RTC_WAKEUP_GetClock
		LL_RTC_WAKEUP_SetClock
	WUTE	LL_RTC_WAKEUP_Disable
		LL_RTC_WAKEUP_Enable
		LL_RTC_WAKEUP_IsEnabled
	WUTIE	LL_RTC_DisableIT_WUT
		LL_RTC_EnableIT_WUT
		LL_RTC_IsEnabledIT_WUT
RTC_DR	DT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	DU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	MT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	MU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	WDU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetWeekDay
		LL_RTC_DATE_SetWeekDay
	YT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear

Register	Field	Function
RTC_DR	YU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
RTC_ICSR	BCDU	LL_RTC_GetBinMixBCDU
		LL_RTC_SetBinMixBCDU
	BIN	LL_RTC_GetBinaryMode
		LL_RTC_SetBinaryMode
	INIT	LL_RTC_DisableInitMode
		LL_RTC_EnableInitMode
	INITF	LL_RTC_IsActiveFlag_INIT
	INITS	LL_RTC_IsActiveFlag_INITS
	RECALPF	LL_RTC_IsActiveFlag_RECALP
	RSF	LL_RTC_ClearFlag_RS
		LL_RTC_IsActiveFlag_RS
	SHPF	LL_RTC_IsActiveFlag_SHP
	WUTWF	LL_RTC_IsActiveFlag_WUTW
RTC_MISR	ALRAMF	LL_RTC_IsActiveFlag_ALRAM
	ALRBMF	LL_RTC_IsActiveFlag_ALRBM
	ITSMF	LL_RTC_IsActiveFlag_ITSM
	SSRUMF	LL_RTC_IsActiveFlag_SSRUM
	TSMF	LL_RTC_IsActiveFlag_TSM
	TSOVMF	LL_RTC_IsActiveFlag_TSOVM
	WUTMF	LL_RTC_IsActiveFlag_WUTM
RTC_PRER	PREDIV_A	LL_RTC_GetAsynchPrescaler
		LL_RTC_SetAsynchPrescaler
	PREDIV_S	LL_RTC_GetSynchPrescaler
		LL_RTC_SetSynchPrescaler
RTC_PRIVCFGR	ALRAPRIV	LL_RTC_SetRtcPrivilege
	ALRBPRIV	LL_RTC_SetRtcPrivilege
	CALPRIV	LL_RTC_SetRtcPrivilege
	INITPRIV	LL_RTC_SetRtcPrivilege
	PRIV	LL_RTC_SetRtcPrivilege
	TSPRIV	LL_RTC_SetRtcPrivilege
	WUTPRIV	LL_RTC_SetRtcPrivilege
RTC_SCR	CALRAF	LL_RTC_ClearFlag_ALRA
	CALRBF	LL_RTC_ClearFlag_ALRB
	CITSF	LL_RTC_ClearFlag_ITS
	CSSRUF	LL_RTC_ClearFlag_SSRU
	CTSF	LL_RTC_ClearFlag_TS
	CTSOVF	LL_RTC_ClearFlag_TSOV

Register	Field	Function
RTC_SCR	CWUTF	LL_RTC_ClearFlag_WUT
RTC_SECCFGR	ALRASEC	LL_RTC_SetRtcSecure
	ALRBSEC	LL_RTC_SetRtcSecure
	CALSEC	LL_RTC_SetRtcSecure
	INISEC	LL_RTC_SetRtcSecure
	INITSEC	LL_RTC_SetRtcSecure
	SEC	LL_RTC_SetRtcSecure
	TSSEC	LL_RTC_SetRtcSecure
	WUTSEC	LL_RTC_SetRtcSecure
RTC_SHIFTR	ADD1S	LL_RTC_TIME_Synchronize
	SUBFS	LL_RTC_TIME_Synchronize
RTC_SR	ALRAF	LL_RTC_IsActiveFlag_ALRA
	ALRBF	LL_RTC_IsActiveFlag_ALRB
	ITSF	LL_RTC_IsActiveFlag_ITS
	SSRUF	LL_RTC_IsActiveFlag_SSRU
	TSF	LL_RTC_IsActiveFlag_TS
	TSOVF	LL_RTC_IsActiveFlag_TSOV
	WUTF	LL_RTC_IsActiveFlag_WUT
RTC_SSR	SS	LL_RTC_TIME_GetSubSecond
RTC_TR	HT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	HU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	MNT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	MNU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	PM	LL_RTC_TIME_Config
		LL_RTC_TIME_GetFormat
		LL_RTC_TIME_SetFormat
	ST	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond

Register	Field	Function
RTC_TR	ST	LL_RTC_TIME_SetSecond
	SU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
RTC_TSDR	DT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	DU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetDay
	MT	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	MU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetMonth
	SS	LL_RTC_TS_GetSubSecond
	WDU	LL_RTC_TS_GetDate
		LL_RTC_TS_GetWeekDay
RTC_TSTR	HT	LL_RTC_TS_GetHour
		LL_RTC_TS_GetTime
	HU	LL_RTC_TS_GetHour
		LL_RTC_TS_GetTime
	MNT	LL_RTC_TS_GetMinute
		LL_RTC_TS_GetTime
	MNU	LL_RTC_TS_GetMinute
		LL_RTC_TS_GetTime
	PM	LL_RTC_TS_GetTimeFormat
	ST	LL_RTC_TS_GetSecond
		LL_RTC_TS_GetTime
	SU	LL_RTC_TS_GetSecond
		LL_RTC_TS_GetTime
RTC_WPR	KEY	LL_RTC_DisableWriteProtection
		LL_RTC_EnableWriteProtection
RTC_WUTR	WUT	LL_RTC_WAKEUP_GetAutoReload
		LL_RTC_WAKEUP_SetAutoReload
TAMP_ATCR1	ATCKSEL	LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
		LL_RTC_TAMPER_ATAMP_SetAsyncPrescaler
	ATOSHARE	LL_RTC_TAMPER_ATAMP_DisableOutputSharing
		LL_RTC_TAMPER_ATAMP_EnableOutputSharing
	ATPER	LL_RTC_TAMPER_ATAMP_GetOutputChangePeriod
		LL_RTC_TAMPER_ATAMP_SetOutputChangePeriod
	FLTEN	LL_RTC_TAMPER_ATAMP_DisableFilter
		LL_RTC_TAMPER_ATAMP_EnableFilter

Register	Field	Function
TAMP_ATCR1	TAMP1AM	LL_RTC_TAMPER_ATAMP_DisableActiveMode
		LL_RTC_TAMPER_ATAMP_EnableActiveMode
	TAMP2AM	LL_RTC_TAMPER_ATAMP_DisableActiveMode
		LL_RTC_TAMPER_ATAMP_EnableActiveMode
	TAMPxAM	LL_RTC_TAMPER_ATAMP_DisableActiveMode
		LL_RTC_TAMPER_ATAMP_EnableActiveMode
TAMP_ATCR2	ATCKSEL	LL_RTC_TAMPER_ATAMP_GetAsyncPrescaler
		LL_RTC_TAMPER_ATAMP_SetSharedOuputSelection
TAMP_ATOR	INITS	LL_RTC_IsActiveFlag_ATAMP_INITS
TAMP_ATSEEDR	SEED	LL_RTC_TAMPER_ATAMP_WriteSeed
TAMP_BKPxR	BKP	LL_RTC_BKP_GetRegister
		LL_RTC_BKP_SetRegister
TAMP_COUNTR	COUNT	LL_RTC_GetMonotonicCounter
		LL_RTC_IncrementMonotonicCounter
TAMP_CR1	ITAMP1E	LL_RTC_TAMPER_ITAMP_Disable
		LL_RTC_TAMPER_ITAMP_Enable
	ITAMP2E	LL_RTC_TAMPER_ITAMP_Disable
	ITAMP2E..	LL_RTC_TAMPER_ITAMP_Enable
	ITAMP3E	LL_RTC_TAMPER_ITAMP_Disable
	ITAMP5E	LL_RTC_TAMPER_ITAMP_Disable
	ITAMP8E	LL_RTC_TAMPER_ITAMP_Disable
	TAMP1E	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
	TAMP2E...	LL_RTC_TAMPER_Disable
		LL_RTC_TAMPER_Enable
TAMP_CR2	TAMP1MF	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP1NOERASE	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP1TRG	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
	TAMP2MF...	LL_RTC_TAMPER_DisableMask
		LL_RTC_TAMPER_EnableMask
	TAMP2NOERASE ...	LL_RTC_TAMPER_DisableEraseBKP
		LL_RTC_TAMPER_EnableEraseBKP
	TAMP2TRG...	LL_RTC_TAMPER_DisableActiveLevel
		LL_RTC_TAMPER_EnableActiveLevel
TAMP_FLTCR	TAMPFLT	LL_RTC_TAMPER_GetFilterCount
		LL_RTC_TAMPER_SetFilterCount
	TAMPFREQ	LL_RTC_TAMPER_GetSamplingFreq
		LL_RTC_TAMPER_SetSamplingFreq

Register	Field	Function
TAMP_FLTCR	TAMPPRCH	LL_RTC_TAMPER_GetPrecharge
		LL_RTC_TAMPER_SetPrecharge
	TAMPPUDIS	LL_RTC_TAMPER_DisablePullUp
		LL_RTC_TAMPER_EnablePullUp
TAMP_IER	ITAMP11IE	LL_RTC_DisableIT_ITAMP11
		LL_RTC_EnableIT_ITAMP11
		LL_RTC_IsEnabledIT_ITAMP11
	ITAMP12IE	LL_RTC_EnableIT_ITAMP12
		LL_RTC_IsEnabledIT_ITAMP12
	ITAMP13IE	LL_RTC_EnableIT_ITAMP13
		LL_RTC_IsEnabledIT_ITAMP13
	ITAMP1IE	LL_RTC_DisableIT_ITAMP1
		LL_RTC_EnableIT_ITAMP1
		LL_RTC_IsEnabledIT_ITAMP1
	ITAMP2IE	LL_RTC_DisableIT_ITAMP2
		LL_RTC_EnableIT_ITAMP2
		LL_RTC_IsEnabledIT_ITAMP2
	ITAMP3IE	LL_RTC_DisableIT_ITAMP3
		LL_RTC_EnableIT_ITAMP3
		LL_RTC_IsEnabledIT_ITAMP3
	ITAMP5IE	LL_RTC_DisableIT_ITAMP5
		LL_RTC_EnableIT_ITAMP5
		LL_RTC_IsEnabledIT_ITAMP5
	ITAMP6IE	LL_RTC_DisableIT_ITAMP6
		LL_RTC_EnableIT_ITAMP6
		LL_RTC_IsEnabledIT_ITAMP6
	ITAMP7IE	LL_RTC_DisableIT_ITAMP7
		LL_RTC_EnableIT_ITAMP7
		LL_RTC_IsEnabledIT_ITAMP7
	ITAMP8IE	LL_RTC_EnableIT_ITAMP8
		LL_RTC_IsEnabledIT_ITAMP8
	ITAMP9IE	LL_RTC_DisableIT_ITAMP9
		LL_RTC_EnableIT_ITAMP9
		LL_RTC_IsEnabledIT_ITAMP9
	TAMP12IE	LL_RTC_DisableIT_ITAMP12
	TAMP13IE	LL_RTC_DisableIT_ITAMP13
	TAMP1IE	LL_RTC_DisableIT_TAMP1
		LL_RTC_EnableIT_TAMP1
		LL_RTC_IsEnabledIT_TAMP1
		LL_RTC_IsEnabledIT_TAMP5
		LL_RTC_IsEnabledIT_TAMP7

Register	Field	Function
TAMP_IER	TAMP2IE	LL_RTC_DisableIT_TAMP2
		LL_RTC_EnableIT_TAMP2
		LL_RTC_IsEnabledIT_TAMP2
	TAMP3IE	LL_RTC_DisableIT_TAMP3
		LL_RTC_EnableIT_TAMP3
		LL_RTC_IsEnabledIT_TAMP3
	TAMP4IE	LL_RTC_DisableIT_TAMP4
		LL_RTC_EnableIT_TAMP4
		LL_RTC_IsEnabledIT_TAMP4
	TAMP5IE	LL_RTC_DisableIT_TAMP5
		LL_RTC_EnableIT_TAMP5
	TAMP6IE	LL_RTC_DisableIT_TAMP6
		LL_RTC_EnableIT_TAMP6
		LL_RTC_IsEnabledIT_TAMP6
	TAMP7IE	LL_RTC_DisableIT_TAMP7
		LL_RTC_EnableIT_TAMP7
	TAMP8IE	LL_RTC_DisableIT_ITAMP8
		LL_RTC_DisableIT_TAMP8
		LL_RTC_EnableIT_TAMP8
		LL_RTC_IsEnabledIT_TAMP8
TAMP_MISR	ITAMP1MF	LL_RTC_IsActiveFlag_ITAMP1M
	ITAMP2MF	LL_RTC_IsActiveFlag_ITAMP2M
	ITAMP3MF	LL_RTC_IsActiveFlag_ITAMP3M
	ITAMP5MF	LL_RTC_IsActiveFlag_ITAMP5M
	ITAMP8MF	LL_RTC_IsActiveFlag_ITAMP8M
	TAMP1MF	LL_RTC_IsActiveFlag_TAMP1M
	TAMP2MF	LL_RTC_IsActiveFlag_TAMP2M
	TAMP3MF	LL_RTC_IsActiveFlag_TAMP3M
		LL_RTC_IsActiveFlag_TAMP6M
	TAMP4MF	LL_RTC_IsActiveFlag_TAMP4M
	TAMP5MF	LL_RTC_IsActiveFlag_TAMP5M
	TAMP7MF	LL_RTC_IsActiveFlag_TAMP7M
	TAMP8MF	LL_RTC_IsActiveFlag_TAMP8M
TAMP_PRIVCFGR	BKPRWPRIV	LL_RTC_SetTampPrivilege
	BKPWPRIV	LL_RTC_SetTampPrivilege
TAMP_SCR	CITAMP1F	LL_RTC_ClearFlag_ITAMP1
	CITAMP2F	LL_RTC_ClearFlag_ITAMP2
	CITAMP3F	LL_RTC_ClearFlag_ITAMP3
	CITAMP5F	LL_RTC_ClearFlag_ITAMP5
	CITAMP8F	LL_RTC_ClearFlag_ITAMP8
	CTAMP1F	LL_RTC_ClearFlag_TAMP1

Register	Field	Function
TAMP_SCR	CTAMP2F	LL_RTC_ClearFlag_TAMP2
	CTAMP3F	LL_RTC_ClearFlag_TAMP3
		LL_RTC_ClearFlag_TAMP4
	CTAMP5F	LL_RTC_ClearFlag_TAMP5
	CTAMP6F	LL_RTC_ClearFlag_TAMP6
	CTAMP7F	LL_RTC_ClearFlag_TAMP7
	CTAMP8F	LL_RTC_ClearFlag_TAMP8
TAMP_SECCFGR	BKPRWSEC	LL_RTC_GetBackupRegProtectionStartZone2
		LL_RTC_SetBackupRegProtection
	BKPWSEC	LL_RTC_GetBackupRegProtectionStartZone3
		LL_RTC_SetBackupRegProtection
	TAMPPRIV	LL_RTC_SetTampPrivilege
	TAMPSEC	LL_RTC_GetTampPrivilege
		LL_RTC_GetTampSecure
		LL_RTC_SetTampSecure
TAMP_SR	ITAMP11F	LL_RTC_IsActiveFlag_ITAMP11
	ITAMP12F	LL_RTC_IsActiveFlag_ITAMP12
	ITAMP13F	LL_RTC_IsActiveFlag_ITAMP13
	ITAMP1F	LL_RTC_IsActiveFlag_ITAMP1
	ITAMP2F	LL_RTC_IsActiveFlag_ITAMP2
	ITAMP3F	LL_RTC_IsActiveFlag_ITAMP3
	ITAMP5F	LL_RTC_IsActiveFlag_ITAMP5
	ITAMP6F	LL_RTC_IsActiveFlag_ITAMP6
	ITAMP7F	LL_RTC_IsActiveFlag_ITAMP7
	ITAMP8F	LL_RTC_IsActiveFlag_ITAMP8
	ITAMP9F	LL_RTC_IsActiveFlag_ITAMP9
	TAMP1F	LL_RTC_IsActiveFlag_TAMP1
	TAMP2F	LL_RTC_IsActiveFlag_TAMP2
	TAMP3F	LL_RTC_IsActiveFlag_TAMP3
	TAMP4F	LL_RTC_IsActiveFlag_TAMP4
	TAMP5F	LL_RTC_IsActiveFlag_TAMP5
	TAMP6F	LL_RTC_IsActiveFlag_TAMP6
	TAMP7F	LL_RTC_IsActiveFlag_TAMP7
	TAMP8F	LL_RTC_IsActiveFlag_TAMP8

124.27 SPI

Table 51. Correspondence between SPI registers and SPI low-layer driver functions

Register	Field	Function
CFG1	BPASS	LL_SPI_GetBaudRatePrescaler
		LL_SPI_SetBaudRatePrescaler

Register	Field	Function
CFG1	CRCEN	LL_SPI_DisableCRC
		LL_SPI_EnableCRC
		LL_SPI_IsEnabledCRC
	CRCSIZE	LL_SPI_GetCRCWidth
		LL_SPI_SetCRCWidth
	DSIZE	LL_SPI_GetDataWidth
		LL_SPI_SetDataWidth
	FTHLV	LL_SPI_GetFIFOThreshold
		LL_SPI_SetFIFOThreshold
	MBR	LL_SPI_GetBaudRatePrescaler
		LL_SPI_SetBaudRatePrescaler
	RXDMAEN	LL_SPI_DisableDMAReq_RX
		LL_SPI_EnableDMAReq_RX
		LL_SPI_IsEnabledDMAReq_RX
	TXDMAEN	LL_SPI_DisableDMAReq_TX
		LL_SPI_EnableDMAReq_TX
		LL_SPI_IsEnabledDMAReq_TX
	UDRCFG	LL_SPI_GetUDRConfiguration
		LL_SPI_SetUDRConfiguration
CFG2	AFCNTR	LL_SPI_DisableGPIOControl
		LL_SPI_EnableGPIOControl
		LL_SPI_IsEnabledGPIOControl
	COMM	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	CPHA	LL_SPI_GetClockPhase
		LL_SPI_SetClockPhase
	CPOL	LL_SPI_GetClockPolarity
		LL_SPI_SetClockPolarity
	IOSWP	LL_SPI_DisableIOSwap
		LL_SPI_EnableIOSwap
		LL_SPI_IsEnabledIOSwap
	LSBFRST	LL_SPI_GetTransferBitOrder
		LL_SPI_SetTransferBitOrder
	MASTER	LL_SPI_GetMode
		LL_SPI_SetMode
	MIDI	LL_SPI_SetInterDataIdleness
	MSSI	LL_SPI_GetMasterSSIdleness
		LL_SPI_SetMasterSSIdleness
	SP	LL_SPI_GetStandard
		LL_SPI_SetStandard
	SSIOP	LL_SPI_GetNSSPolarity

Register	Field	Function
CFG2	SSIOP	LL_SPI_SetNSSPolarity
	SSM	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
	SSOE	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
	SSOM	LL_SPI_DisableNSSPulseMgt
		LL_SPI_EnableNSSPulseMgt
		LL_SPI_IsEnabledNSSPulse
CR1	CRC33_17	LL_SPI_DisableFullSizeCRC
		LL_SPI_EnableFullSizeCRC
		LL_SPI_IsEnabledFullSizeCRC
	CSTART	LL_SPI_IsActiveMasterTransfer
		LL_SPI_StartMasterTransfer
	CSUSP	LL_SPI_SuspendMasterTransfer
	HDDIR	LL_SPI_GetHalfDuplexDirection
		LL_SPI_GetTransferDirection
		LL_SPI_SetHalfDuplexDirection
		LL_SPI_SetTransferDirection
	IOLOCK	LL_SPI_EnableIOLock
		LL_SPI_IsEnabledIOLock
	MASRX	LL_SPI_DisableMasterRxAutoSuspend
		LL_SPI_EnableMasterRxAutoSuspend
		LL_SPI_IsEnabledMasterRxAutoSuspend
	RCRCINI	LL_SPI_GetRxCRCInitPattern
		LL_SPI_SetRxCRCInitPattern
	SPE	LL_SPI_Disable
		LL_SPI_Enable
		LL_SPI_IsEnabled
	SSI	LL_SPI_GetInternalSSLevel
		LL_SPI_SetInternalSSLevel
	TCRCINI	LL_SPI_GetTxCRCInitPattern
		LL_SPI_SetTxCRCInitPattern
CR2	TSIZE	LL_SPI_GetTransferSize
		LL_SPI_SetTransferSize
CRCPOLY	CRCPOLY	LL_SPI_GetCRCPolynomial
		LL_SPI_SetCRCPolynomial
IER	CRCEIE	LL_SPI_DisableIT_CRCERR
		LL_SPI_EnableIT_CRCERR
		LL_SPI_IsEnabledIT_CRCERR
	DXPIE	LL_SPI_DisableIT_DXP
		LL_SPI_EnableIT_DXP

Register	Field	Function
IER	DXPIE	LL_SPI_IsEnabledIT_DXP
	EOTIE	LL_SPI_DisableIT_EOT
		LL_SPI_EnableIT_EOT
		LL_SPI_IsEnabledIT_EOT
	MODFIE	LL_SPI_DisableIT_MODF
		LL_SPI_EnableIT_MODF
		LL_SPI_IsEnabledIT_MODF
	OVRIE	LL_SPI_DisableIT_OVR
		LL_SPI_EnableIT_OVR
		LL_SPI_IsEnabledIT_OVR
	RXPIE	LL_SPI_DisableIT_RXP
		LL_SPI_EnableIT_RXP
		LL_SPI_IsEnabledIT_RXP
	TIFREIE	LL_SPI_DisableIT_FRE
		LL_SPI_EnableIT_FRE
		LL_SPI_IsEnabledIT_FRE
	TXPIE	LL_SPI_DisableIT_TXP
		LL_SPI_EnableIT_TXP
		LL_SPI_IsEnabledIT_TXP
	TXTFIE	LL_SPI_DisableIT_TXTF
		LL_SPI_EnableIT_TXTF
		LL_SPI_IsEnabledIT_TXTF
	UDRIE	LL_SPI_DisableIT_UDR
		LL_SPI_EnableIT_UDR
		LL_SPI_IsEnabledIT_UDR
IFCR	CRCEC	LL_SPI_ClearFlag_CRCERR
	EOTC	LL_SPI_ClearFlag_EOT
	MODFC	LL_SPI_ClearFlag_MODF
	OVRC	LL_SPI_ClearFlag_OVR
	SUSPC	LL_SPI_ClearFlag_SUSP
	TIFREC	LL_SPI_ClearFlag_FRE
	TXTFC	LL_SPI_ClearFlag_TXTF
	UDRC	LL_SPI_ClearFlag_UDR
RXCRCR	RXCRC	LL_SPI_GetRxCRC
RXDR	.	LL_SPI_ReceiveData16
		LL_SPI_ReceiveData32
		LL_SPI_ReceiveData8
	RXDR	LL_SPI_DMA_GetRxRegAddr
SR	CRCE	LL_SPI_IsActiveFlag_CRCERR
	CTSIZE	LL_SPI_GetRemainingDataFrames
	DXP	LL_SPI_IsActiveFlag_DXP

Register	Field	Function
SR	EOT	LL_SPI_IsActiveFlag_EOT
	MODF	LL_SPI_IsActiveFlag_MODF
	OVR	LL_SPI_IsActiveFlag_OVR
	RXP	LL_SPI_IsActiveFlag_RXP
	RXPLVL	LL_SPI_GetRxFIFOPackingLevel
	RXWNE	LL_SPI_IsActiveFlag_RXWNE
	SUSP	LL_SPI_IsActiveFlag_SUSP
	TIFRE	LL_SPI_IsActiveFlag_FRE
	TXC	LL_SPI_IsActiveFlag_TXC
	TXP	LL_SPI_IsActiveFlag_TXP
	TXTF	LL_SPI_IsActiveFlag_TXTF
	UDR	LL_SPI_IsActiveFlag_UDR
TXCRCR	TXCRC	LL_SPI_GetTxCRC
TXDR	.	LL_SPI_TransmitData16
		LL_SPI_TransmitData32
		LL_SPI_TransmitData8
	TXDR	LL_SPI_DMA_GetTxRegAddr
UDRDR	UDRDR	LL_SPI_GetUDRPattern
		LL_SPI_SetUDRPattern

124.28 SYSTEM

Table 52. **Correspondence between SYSTEM registers and SYSTEM low-layer driver functions**

Register	Field	Function
CCCR	NCC1	LL_SYSCFG_GetNMOSVddCompensationCode
	NCC2	LL_SYSCFG_GetNMOSVddIO2CompensationCode
		LL_SYSCFG_SetNMOSVddIO2CompensationCode
	NCC3	LL_SYSCFG_GetNMOSVddHSPICompensationCode
		LL_SYSCFG_SetNMOSVddHSPICompensationCode
	PCC1	LL_SYSCFG_GetPMOSVddCompensationCode
		LL_SYSCFG_SetPMOSVddCompensationCode
	PCC2	LL_SYSCFG_GetPMOSVddIO2CompensationCode
		LL_SYSCFG_SetNMOSVddCompensationCode
		LL_SYSCFG_SetPMOSVddIO2CompensationCode
	PCC3	LL_SYSCFG_GetPMOSVddHSPICompensationCode
		LL_SYSCFG_SetPMOSVddHSPICompensationCode
CCCSR	CS1	LL_SYSCFG_GetVddCellCompensationCode
		LL_SYSCFG_SetVddCellCompensationCode
	CS2	LL_SYSCFG_GetVddIO2CellCompensationCode
		LL_SYSCFG_SetVddIO2CellCompensationCode
	CS3	LL_SYSCFG_GetVddHSPICellCompensationCode

Register	Field	Function
CCCSR	CS3	LL_SYSCFG_SetVddHSPICellCompensationCode
	EN1	LL_SYSCFG_DisableVddCompensationCell
		LL_SYSCFG_EnableVddCompensationCell
		LL_SYSCFG_IsEnabled_VddCompensationCell
	EN2	LL_SYSCFG_DisableVddIO2CompensationCell
		LL_SYSCFG_EnableVddIO2CompensationCell
		LL_SYSCFG_IsEnabled_VddIO2CompensationCell
	EN3	LL_SYSCFG_DisableVddHSPICompensationCell
		LL_SYSCFG_EnableVddHSPICompensationCell
		LL_SYSCFG_IsEnabled_VddHSPICompensationCell
	RDY1	LL_SYSCFG_IsActiveFlag_VddCMPCR
	RDY2	LL_SYSCFG_IsActiveFlag_VddIO2CMPCR
	RDY3	LL_SYSCFG_IsActiveFlag_VddHSPICMPCR
CCVR	NCV1	LL_SYSCFG_GetNMOSVddCompensationValue
	NCV2	LL_SYSCFG_GetNMOSVddIO2CompensationValue
	NCV3	LL_SYSCFG_GetNMOSVddHSPICompensationValue
	PCV1	LL_SYSCFG_GetPMOSVddCompensationValue
	PCV2	LL_SYSCFG_GetPMOSVddIO2CompensationValue
	PCV3	LL_SYSCFG_GetPMOSVddHSPICompensationValue
DBGMCU_AHB1FZ	DBG_GPDMAx_STOP	LL_DBGMCU_AHB1_GRP1_FreezePeriph
DBGMCU_AHB3FZ	DBG_LPDMAx_STOP	LL_DBGMCU_AHB3_GRP1_FreezePeriph
DBGMCU_APB1FZR1	DBG_xxxx_STOP	LL_DBGMCU_APB1_GRP1_FreezePeriph
		LL_DBGMCU_APB1_GRP1_UnFreezePeriph
DBGMCU_APB1FZR2	DBG_xxxx_STOP	LL_DBGMCU_APB1_GRP2_FreezePeriph
		LL_DBGMCU_APB1_GRP2_UnFreezePeriph
DBGMCU_APB2FZ	DBG_TIMx_STOP	LL_DBGMCU_APB2_GRP1_FreezePeriph
DBGMCU_APB2FZR	DBG_TIMx_STOP	LL_DBGMCU_APB2_GRP1_UnFreezePeriph
DBGMCU_APB3FZ	DBG_TIMx_STOP	LL_DBGMCU_APB3_GRP1_FreezePeriph
DBGMCU_APB3FZR	DBG_TIMx_STOP	LL_DBGMCU_APB3_GRP1_UnFreezePeriph
DBGMCU_CR	DBG_STANDBY	LL_DBGMCU_DisableDBGStandbyMode
		LL_DBGMCU_EnableDBGStandbyMode
	DBG_STOP	LL_DBGMCU_DisableDBGStopMode
		LL_DBGMCU_EnableDBGStopMode
	TRACE_CLKEN	LL_DBGMCU_DisableTraceClock
		LL_DBGMCU_EnableTraceClock
	TRACE_IOEN	LL_DBGMCU_GetTracePinAssignment
		LL_DBGMCU_SetTracePinAssignment
	TRACE_MODE	LL_DBGMCU_GetTracePinAssignment
		LL_DBGMCU_SetTracePinAssignment
DBGMCU_IDCODE	DEV_ID	LL_DBGMCU_GetDeviceID
	REV_ID	LL_DBGMCU_GetRevisionID

Register	Field	Function
FLASH_ACR	LATENCY	LL_FLASH_GetLatency
		LL_FLASH_SetLatency
	RUN_PD	LL_FLASH_EnableRunPowerDown
	SLEEP_PD	LL_FLASH_DisableSleepPowerDown
		LL_FLASH_EnableSleepPowerDown
FLASH_PDKEYR	PDKEY1_1	LL_FLASH_EnableRunPowerDown
	PDKEY1_2	LL_FLASH_EnableRunPowerDown
	PDKEY2_1	LL_FLASH_EnableRunPowerDown
	PDKEY2_2	LL_FLASH_EnableRunPowerDown
MESR	IPMEE	LL_SYSCFG_ClearEraseEndStatus
		LL_SYSCFG_GetEraseEndStatus
	MCLR	LL_SYSCFG_ClearEraseAfterResetStatus
		LL_SYSCFG_GetEraseAfterResetStatus
SECCFGR	CLASSBSEC	LL_SYSCFG_ConfigSecure
	FPUSEC	LL_SYSCFG_ConfigSecure
	SYSCFGSEC	LL_SYSCFG_ConfigSecure
SYSCFG_CFGR1	ANASWVDD	LL_SYSCFG_DisableAnalogSwitchVdd
		LL_SYSCFG_EnableAnalogSwitchVdd
	BOOSTEN	LL_SYSCFG_DisableAnalogBooster
		LL_SYSCFG_EnableAnalogBooster
	PBx_FMP	LL_SYSCFG_DisableFastModePlus
		LL_SYSCFG_EnableFastModePlus
SYSCFG_CFGR2	CLL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
	ECCL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
	PVDL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
	SPL	LL_SYSCFG_GetTIMBreakInputs
		LL_SYSCFG_SetTIMBreakInputs
SYSCFG_FPUIMR	FPU_IE_0	LL_SYSCFG_DisableIT_FPU_IOC
		LL_SYSCFG_EnableIT_FPU_IOC
		LL_SYSCFG_IsEnabledIT_FPU_IOC
	FPU_IE_1	LL_SYSCFG_DisableIT_FPU_DZC
		LL_SYSCFG_EnableIT_FPU_DZC
		LL_SYSCFG_IsEnabledIT_FPU_DZC
	FPU_IE_2	LL_SYSCFG_DisableIT_FPU_UFC
		LL_SYSCFG_EnableIT_FPU_UFC
		LL_SYSCFG_IsEnabledIT_FPU_UFC
	FPU_IE_3	LL_SYSCFG_DisableIT_FPU_OFC
		LL_SYSCFG_EnableIT_FPU_OFC

Register	Field	Function
SYSCFG_FPUIMR	FPU_IE_3	LL_SYSCFG_IsEnabledIT_FPU_OFC
	FPU_IE_4	LL_SYSCFG_DisableIT_FPU_IDC
		LL_SYSCFG_EnableIT_FPU_IDC
		LL_SYSCFG_IsEnabledIT_FPU_IDC
	FPU_IE_5	LL_SYSCFG_DisableIT_FPU_IXC
		LL_SYSCFG_EnableIT_FPU_IXC
		LL_SYSCFG_IsEnabledIT_FPU_IXC
VREFBUF_CCR	TRIM	LL_VREFBUF_GetTrimming
		LL_VREFBUF_SetTrimming
VREFBUF_CSR	ENVR	LL_VREFBUF_Disable
		LL_VREFBUF_Enable
	HIZ	LL_VREFBUF_DisableHIZ
		LL_VREFBUF_EnableHIZ
	VRR	LL_VREFBUF_IsVREFReady
	VRS	LL_VREFBUF_GetVoltageScaling
		LL_VREFBUF_SetVoltageScaling

124.29 TIM

Table 53. **Correspondence between TIM registers and TIM low-layer driver functions**

Register	Field	Function
AF1	BKCMP1E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKCMP1P	LL_TIM_SetBreakInputSourcePolarity
	BKCMP2E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKCMP2P	LL_TIM_SetBreakInputSourcePolarity
	BKDF1BK0E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKINE	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKINP	LL_TIM_SetBreakInputSourcePolarity
	ETRSEL	LL_TIM_SetETRSource
AF2	BK2CMP1E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BK2CMP1P	LL_TIM_SetBreakInputSourcePolarity
	BK2CMP2E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BK2CMP2P	LL_TIM_SetBreakInputSourcePolarity
	BK2DF1BK1E	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource

Register	Field	Function
AF2	BK2INE	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BK2INP	LL_TIM_SetBreakInputSourcePolarity
	OCRSEL	LL_TIM_SetOCRefClearInputSource
ARR	ARR	LL_TIM_GetAutoReload
		LL_TIM_SetAutoReload
BDTR	AOE	LL_TIM_DisableAutomaticOutput
		LL_TIM_EnableAutomaticOutput
		LL_TIM_IsEnabledAutomaticOutput
	BK2BID	LL_TIM_ConfigBRK2
	BK2DSRM	LL_TIM_DisarmBRK2
	BK2E	LL_TIM_DisableBRK2
		LL_TIM_EnableBRK2
	BK2F	LL_TIM_ConfigBRK2
	BK2P	LL_TIM_ConfigBRK2
	BKBID	LL_TIM_ConfigBRK
	BKDSRM	LL_TIM_DisarmBRK
	BKE	LL_TIM_DisableBRK
		LL_TIM_EnableBRK
	BKF	LL_TIM_ConfigBRK
	BKP	LL_TIM_ConfigBRK
	DTG	LL_TIM_OC_SetDeadTime
	LOCK	LL_TIM_CC_SetLockLevel
	MOE	LL_TIM_DisableAllOutputs
		LL_TIM_EnableAllOutputs
		LL_TIM_IsEnabledAllOutputs
	OSSI	LL_TIM_SetOffStates
	OSSR	LL_TIM_SetOffStates
CCER	CC1E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC1P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity

Register	Field	Function
CCER	CC1P	LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel

Register	Field	Function
CCER	CC4NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC4NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC5E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC5P	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC6E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC6P	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
CCMR1	CC1S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC2S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC1F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC1PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC2F	LL_TIM_IC_Config

Register	Field	Function
CCMR1	IC2F	LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC2PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC1CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC1FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC1M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC1PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC2CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC2FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC2M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC2PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR2	CC3S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC4S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC3F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC3PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler

Register	Field	Function
CCMR2	IC4F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC4PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	OC3CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC3FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC3M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC3PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC4CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC4FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC4PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR3	CC5S	LL_TIM_OC_ConfigOutput
	CC6S	LL_TIM_OC_ConfigOutput
	OC5CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC5FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC5M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC5PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload

Register	Field	Function
CCMR3	OC6CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC6FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC6M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC6PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCR1	CCR1	LL_TIM_IC_GetCaptureCH1
		LL_TIM_OC_GetCompareCH1
		LL_TIM_OC_SetCompareCH1
CCR2	CCR2	LL_TIM_IC_GetCaptureCH2
		LL_TIM_OC_GetCompareCH2
		LL_TIM_OC_SetCompareCH2
CCR3	CCR3	LL_TIM_IC_GetCaptureCH3
		LL_TIM_OC_GetCompareCH3
		LL_TIM_OC_SetCompareCH3
CCR4	CCR4	LL_TIM_IC_GetCaptureCH4
		LL_TIM_OC_GetCompareCH4
		LL_TIM_OC_SetCompareCH4
CCR5	CCR5	LL_TIM_OC_GetCompareCH5
		LL_TIM_OC_SetCompareCH5
	GC5C1	LL_TIM_SetCH5CombinedChannels
	GC5C2	LL_TIM_SetCH5CombinedChannels
	GC5C3	LL_TIM_SetCH5CombinedChannels
CCR6	CCR6	LL_TIM_OC_GetCompareCH6
		LL_TIM_OC_SetCompareCH6
CNT	CNT	LL_TIM_GetCounter
		LL_TIM_SetCounter
CR1	ARPE	LL_TIM_DisableARRPreload
		LL_TIM_EnableARRPreload
		LL_TIM_IsEnabledARRPreload
	CEN	LL_TIM_DisableCounter
		LL_TIM_EnableCounter
		LL_TIM_IsEnabledCounter
	CKD	LL_TIM_GetClockDivision
		LL_TIM_SetClockDivision
	CMS	LL_TIM_GetCounterMode

Register	Field	Function
CR1	CMS	LL_TIM_SetCounterMode
	DIR	LL_TIM_GetCounterMode
		LL_TIM_GetDirection
		LL_TIM_SetCounterMode
	DITHEN	LL_TIM_DisableDithering
		LL_TIM_EnableDithering
		LL_TIM_IsEnabledDithering
	OPM	LL_TIM_GetOnePulseMode
		LL_TIM_SetOnePulseMode
	UDIS	LL_TIM_DisableUpdateEvent
		LL_TIM_EnableUpdateEvent
		LL_TIM_IsEnabledUpdateEvent
	UIFREMAP	LL_TIM_DisableUIFRemap
		LL_TIM_EnableUIFRemap
	URS	LL_TIM_GetUpdateSource
		LL_TIM_SetUpdateSource
CR2	CCDS	LL_TIM_CC_GetDMAReqTrigger
		LL_TIM_CC_SetDMAReqTrigger
	CCPC	LL_TIM_CC_DisablePreload
		LL_TIM_CC_EnablePreload
		LL_TIM_CC_IsEnabledPreload
	CCUS	LL_TIM_CC_SetUpdate
	MMS	LL_TIM_SetTriggerOutput
	MMS2	LL_TIM_SetTriggerOutput2
	OIS1	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS4	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS4N	LL_TIM_OC_GetIdleState

Register	Field	Function
CR2	OIS4N	LL_TIM_OC_SetIdleState
	OIS5	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS6	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	TI1S	LL_TIM_IC_DisableXORCombination
		LL_TIM_IC_EnableXORCombination
		LL_TIM_IC_IsEnabledXORCombination
DCR	DBA	LL_TIM_ConfigDMABurst
	DBL	LL_TIM_ConfigDMABurst
DIER	BIE	LL_TIM_DisableIT_BRK
		LL_TIM_EnableIT_BRK
		LL_TIM_IsEnabledIT_BRK
	CC1DE	Section 119.2.1: LL_LPTIM_DisableDMAReq_CC1
		Section 119.2.1: LL_LPTIM_EnableDMAReq_CC1
		Section 119.2.1: LL_LPTIM_IsEnabledDMAReq_CC1
		LL_TIM_DisableDMAReq_CC1
		LL_TIM_EnableDMAReq_CC1
		LL_TIM_IsEnabledDMAReq_CC1
	CC1IE	LL_TIM_DisableIT_CC1
		LL_TIM_EnableIT_CC1
		LL_TIM_IsEnabledIT_CC1
	CC2DE	Section 119.2.1: LL_LPTIM_DisableDMAReq_CC2
		Section 119.2.1: LL_LPTIM_EnableDMAReq_CC2
		Section 119.2.1: LL_LPTIM_IsEnabledDMAReq_CC2
		LL_TIM_DisableDMAReq_CC2
		LL_TIM_EnableDMAReq_CC2
		LL_TIM_IsEnabledDMAReq_CC2
	CC2IE	LL_TIM_DisableIT_CC2
		LL_TIM_EnableIT_CC2
		LL_TIM_IsEnabledIT_CC2
	CC3DE	LL_TIM_DisableDMAReq_CC3
		LL_TIM_EnableDMAReq_CC3
		LL_TIM_IsEnabledDMAReq_CC3
	CC3IE	LL_TIM_DisableIT_CC3
		LL_TIM_EnableIT_CC3
		LL_TIM_IsEnabledIT_CC3
	CC4DE	LL_TIM_DisableDMAReq_CC4
		LL_TIM_EnableDMAReq_CC4

Register	Field	Function
DIER	CC4DE	LL_TIM_IsEnabledDMAReq_CC4
	CC4IE	LL_TIM_DisableIT_CC4
		LL_TIM_EnableIT_CC4
		LL_TIM_IsEnabledIT_CC4
	COMDE	LL_TIM_DisableDMAReq_COM
		LL_TIM_EnableDMAReq_COM
		LL_TIM_IsEnabledDMAReq_COM
	COMIE	LL_TIM_DisableIT_COM
		LL_TIM_EnableIT_COM
		LL_TIM_IsEnabledIT_COM
	DIRIE	LL_TIM_DisableIT_DIR
		LL_TIM_EnableIT_DIR
		LL_TIM_IsEnabledIT_DIR
	IDXIE	LL_TIM_DisableIT_IDX
		LL_TIM_EnableIT_IDX
		LL_TIM_IsEnabledIT_IDX
	IERRIE	LL_TIM_DisableIT_IERR
		LL_TIM_EnableIT_IERR
		LL_TIM_IsEnabledIT_IERR
	TDE	LL_TIM_DisableDMAReq_TRIG
		LL_TIM_EnableDMAReq_TRIG
		LL_TIM_IsEnabledDMAReq_TRIG
	TERRIE	LL_TIM_DisableIT_TERR
		LL_TIM_EnableIT_TERR
		LL_TIM_IsEnabledIT_TERR
	TIE	LL_TIM_DisableIT_TRIG
		LL_TIM_EnableIT_TRIG
		LL_TIM_IsEnabledIT_TRIG
	UDE	LL_TIM_DisableDMAReq_UPDATE
		LL_TIM_EnableDMAReq_UPDATE
		LL_TIM_IsEnabledDMAReq_UPDATE
	UEDE	Section 119.2.1: LL_LPTIM_DisableDMAReq_UPDATE
		Section 119.2.1: LL_LPTIM_EnableDMAReq_UPDATE
		Section 119.2.1: LL_LPTIM_IsEnabledDMAReq_UPDATE
	UIE	LL_TIM_DisableIT_UPDATE
		LL_TIM_EnableIT_UPDATE
		LL_TIM_IsEnabledIT_UPDATE
DTR2	DTAE	LL_TIM_DisableAsymmetricalDeadTime
		LL_TIM_EnableAsymmetricalDeadTime
		LL_TIM_IsEnabledAsymmetricalDeadTime
	DTGF	LL_TIM_GetFallingDeadTime

Register	Field	Function
DTR2	DTGF	LL_TIM_SetFallingDeadTime
	DTPE	LL_TIM_DisableDeadTimePreload
		LL_TIM_EnableDeadTimePreload
		LL_TIM_IsEnabledDeadTimePreload
ECR	FIDX	LL_TIM_ConfigIDX
		LL_TIM_DisableFirstIndex
		LL_TIM_EnableFirstIndex
		LL_TIM_IsEnabledFirstIndex
	IBLK	LL_TIM_ConfigIDX
		LL_TIM_GetIndexblanking
		LL_TIM_SetIndexblanking
	IDIR	LL_TIM_ConfigIDX
		LL_TIM_GetIndexDirection
		LL_TIM_SetIndexDirection
	IE	LL_TIM_DisableEncoderIndex
		LL_TIM_EnableEncoderIndex
		LL_TIM_IsEnabledEncoderIndex
	IPOS	LL_TIM_ConfigIDX
		LL_TIM_GetIndexPositionning
		LL_TIM_SetIndexPositionning
	PW	LL_TIM_OC_GetPulseWidth
		LL_TIM_OC_SetPulseWidth
	PWPRSC	LL_TIM_OC_GetPulseWidthPrescaler
		LL_TIM_OC_SetPulseWidthPrescaler
EGR	B2G	LL_TIM_GenerateEvent_BRK2
	BG	LL_TIM_GenerateEvent_BRK
	CC1G	LL_TIM_GenerateEvent_CC1
	CC2G	LL_TIM_GenerateEvent_CC2
	CC3G	LL_TIM_GenerateEvent_CC3
	CC4G	LL_TIM_GenerateEvent_CC4
	COMG	LL_TIM_GenerateEvent_COM
	TG	LL_TIM_GenerateEvent_TRIG
	UG	LL_TIM_GenerateEvent_UPDATE
OR1	HSE32EN	LL_TIM_DisableHSE32
		LL_TIM_EnableHSE32
		LL_TIM_IsEnabledHSE32
PSC	PSC	LL_TIM_GetPrescaler
		LL_TIM_SetPrescaler
RCR	REP	LL_TIM_GetRepetitionCounter
		LL_TIM_SetRepetitionCounter
SMCR	ECE	LL_TIM_DisableExternalClock

Register	Field	Function
SMCR	ECE	LL_TIM_EnableExternalClock
		LL_TIM_IsEnabledExternalClock
		LL_TIM_SetClockSource
	ETF	LL_TIM_ConfigETR
	ETP	LL_TIM_ConfigETR
	ETPS	LL_TIM_ConfigETR
	MSM	LL_TIM_DisableMasterSlaveMode
		LL_TIM_EnableMasterSlaveMode
		LL_TIM_IsEnabledMasterSlaveMode
	OCCS	LL_TIM_SetOCRefClearInputSource
	SMS	LL_TIM_SetClockSource
		LL_TIM_SetEncoderMode
		LL_TIM_SetSlaveMode
	SMSPE	LL_TIM_DisableSMSPreload
		LL_TIM_EnableSMSPreload
		LL_TIM_IsEnabledSMSPreload
	SMSPS	LL_TIM_GetSMSPreloadSource
		LL_TIM_SetSMSPreloadSource
	TS	LL_TIM_SetTriggerInput
SR	B2IF	LL_TIM_ClearFlag_BRK2
		LL_TIM_IsActiveFlag_BRK2
	BIF	LL_TIM_ClearFlag_BRK
		LL_TIM_IsActiveFlag_BRK
	CC1IF	LL_TIM_ClearFlag_CC1
		LL_TIM_IsActiveFlag_CC1
	CC1OF	LL_TIM_ClearFlag_CC1OVR
		LL_TIM_IsActiveFlag_CC1OVR
	CC2IF	LL_TIM_ClearFlag_CC2
		LL_TIM_IsActiveFlag_CC2
	CC2OF	LL_TIM_ClearFlag_CC2OVR
		LL_TIM_IsActiveFlag_CC2OVR
	CC3IF	LL_TIM_ClearFlag_CC3
		LL_TIM_IsActiveFlag_CC3
	CC3OF	LL_TIM_ClearFlag_CC3OVR
		LL_TIM_IsActiveFlag_CC3OVR
	CC4IF	LL_TIM_ClearFlag_CC4
		LL_TIM_IsActiveFlag_CC4
	CC4OF	LL_TIM_ClearFlag_CC4OVR
		LL_TIM_IsActiveFlag_CC4OVR
	CC5IF	LL_TIM_ClearFlag_CC5
		LL_TIM_IsActiveFlag_CC5

Register	Field	Function
SR	CC6IF	LL_TIM_ClearFlag_CC6
		LL_TIM_IsActiveFlag_CC6
	COMIF	LL_TIM_ClearFlag_COM
		LL_TIM_IsActiveFlag_COM
	DIRF	LL_TIM_ClearFlag_DIR
		LL_TIM_IsActiveFlag_DIR
	IDXF	LL_TIM_ClearFlag_IDX
		LL_TIM_IsActiveFlag_IDX
	IERRF	LL_TIM_ClearFlag_IERR
		LL_TIM_IsActiveFlag_IERR
	SBIF	LL_TIM_ClearFlag_SYSBRK
		LL_TIM_IsActiveFlag_SYSBRK
	TERRF	LL_TIM_ClearFlag_TERR
		LL_TIM_IsActiveFlag_TERR
	TIF	LL_TIM_ClearFlag_TRIG
		LL_TIM_IsActiveFlag_TRIG
	UIF	LL_TIM_ClearFlag_UPDATE
		LL_TIM_IsActiveFlag_UPDATE
TIM15_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM16_TISEL	TI1SEL	LL_TIM_SetRemap
TIM17_TISEL	TI1SEL	LL_TIM_SetRemap
TIM1_TISEL	TI1SEL	LL_TIM_SetRemap
TIM2_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
	TI4SEL	LL_TIM_SetRemap
TIM3_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM4_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM5_TISEL	TI1SEL	LL_TIM_SetRemap
	TI2SEL	LL_TIM_SetRemap
TIM8_TISEL	TI1SEL	LL_TIM_SetRemap

124.30 UCPD

Table 54. Correspondence between UCPD registers and UCPD low-layer driver functions

Register	Field	Function
CFG1	HBITCLKDIV	LL_UCPD_SetHbitClockDiv
	IFRGAP	LL_UCPD_SetIfrGap
	RXDMAEN	LL_UCPD_RxDMADisable

Register	Field	Function
CFG1	RXDMAEN	LL_UCPD_RxDMAEnable
	RXORDSETEN	LL_UCPD_SetRxOrderSet
	TRANSWIN	LL_UCPD_SetTransWin
	TXDMAEN	LL_UCPD_TxDMADisable
		LL_UCPD_TxDMAEnable
	UCPDCLK	LL_UCPD_SetPSCClk
	UCPDEN	LL_UCPD_Disable
		LL_UCPD_Enable
		LL_UCPD_IsEnabled
CFG2	FORCECLK	LL_UCPD_ForceClockDisable
		LL_UCPD_ForceClockEnable
	RXAFILTEN	LL_UCPD_RxAnalogFilterDisable
		LL_UCPD_RxAnalogFilterEnable
	RXFILTDIS	LL_UCPD_RxFilterDisable
		LL_UCPD_RxFilterEnable
	WUPEN	LL_UCPD_WakeUpDisable
		LL_UCPD_WakeUpEnable
CR	ANAMODE	LL_UCPD_GetRole
		LL_UCPD_SetSNKRole
		LL_UCPD_SetSRCRole
	ANASUBMODE	LL_UCPD_SetRpResistor
	CC1TCDIS	LL_UCPD_TypeCDetectionCC1Disable
		LL_UCPD_TypeCDetectionCC1Enable
	CC1VCONNEN	LL_UCPD_SetccEnable
	CC2TCDIS	LL_UCPD_TypeCDetectionCC2Disable
		LL_UCPD_TypeCDetectionCC2Enable
	FRSRXEN	LL_UCPD_FRSDetectionDisable
		LL_UCPD_FRSDetectionEnable
	FRSTX	LL_UCPD_VconnDischargeDisable
	PHYCCSEL	LL_UCPD_SetCCPin
	PHYRXEN	LL_UCPD_RxDisable
		LL_UCPD_RxEnable
	RDCH	LL_UCPD_VconnDischargeDisable
		LL_UCPD_VconnDischargeEnable
	RXMODE	LL_UCPD_SetRxMode
	TXHRST	LL_UCPD_SendHardReset
	TXMODE	LL_UCPD_SetTxMode
	TXSEND	LL_UCPD_SendMessage
CR2	RXDMAEN	LL_UCPD_IsEnabledRxDMA
	TXDMAEN	LL_UCPD_IsEnabledTxDMA
ICR	FRSEVTIE	LL_UCPD_ClearFlag_FRS

Register	Field	Function
ICR	HRSTDISCIE	LL_UCPD_ClearFlag_TxHRSTDISC
	HRSTSENTIE	LL_UCPD_ClearFlag_TxHRSTSENT
	RXHRSTDETIE	LL_UCPD_ClearFlag_RxHRST
	RXMSGENDIE	LL_UCPD_ClearFlag_RxMsgEnd
	RXORDDETIE	LL_UCPD_ClearFlag_RxOrderSet
	RXOVRIE	LL_UCPD_ClearFlag_RxOvr
	TXMSGABTIE	LL_UCPD_ClearFlag_TxMSGABT
	TXMSGDISCIE	LL_UCPD_ClearFlag_TxMSGDISC
	TXMSGSENTIE	LL_UCPD_ClearFlag_TxMSGSENT
	TXUNDIE	LL_UCPD_ClearFlag_TxUND
IIMR	TYPECEVT1IE	LL_UCPD_ClearFlag_TypeCEventCC1
	TYPECEVT2IE	LL_UCPD_ClearFlag_TypeCEventCC2
IMR	FRSEVTIE	LL_UCPD_DisableIT_FRS
		LL_UCPD_EnableIT_FRS
	HRSTDISCIE	LL_UCPD_DisableIT_TxHRSTDISC
		LL_UCPD_EnableIT_TxHRSTDISC
		LL_UCPD_IsEnableIT_TxHRSTDISC
	HRSTSENTIE	LL_UCPD_DisableIT_TxHRSTSENT
		LL_UCPD_EnableIT_TxHRSTSENT
		LL_UCPD_IsEnableIT_TxHRSTSENT
	RXHRSTDETIE	LL_UCPD_DisableIT_RxHRST
		LL_UCPD_EnableIT_RxHRST
		LL_UCPD_IsEnableIT_RxHRST
	RXMSGENDIE	LL_UCPD_DisableIT_RxMsgEnd
		LL_UCPD_EnableIT_RxMsgEnd
		LL_UCPD_IsEnableIT_RxMsgEnd
	RXNEIE	LL_UCPD_DisableIT_RxNE
		LL_UCPD_EnableIT_RxNE
		LL_UCPD_IsEnableIT_RxNE
	RXORDDETIE	LL_UCPD_DisableIT_RxOrderSet
		LL_UCPD_EnableIT_RxOrderSet
		LL_UCPD_IsEnableIT_RxOrderSet
	RXOVRIE	LL_UCPD_DisableIT_RxOvr
		LL_UCPD_EnableIT_RxOvr
		LL_UCPD_IsEnableIT_RxOvr
	TXISIE	LL_UCPD_DisableIT_TxIS
		LL_UCPD_EnableIT_TxIS
		LL_UCPD_IsEnableIT_TxIS
	TXMSGABTIE	LL_UCPD_DisableIT_TxMSGABT
		LL_UCPD_EnableIT_TxMSGABT
		LL_UCPD_IsEnableIT_TxMSGABT

Register	Field	Function
IMR	TXMSGDISCIE	LL_UCPD_DisableIT_TxMSGDISC
		LL_UCPD_EnableIT_TxMSGDISC
		LL_UCPD_IsEnableIT_TxMSGDISC
	TXMSGSENTIE	LL_UCPD_DisableIT_TxMSGSENT
		LL_UCPD_EnableIT_TxMSGSENT
		LL_UCPD_IsEnableIT_TxMSGSENT
	TXUNDIE	LL_UCPD_DisableIT_TxUND
		LL_UCPD_EnableIT_TxUND
		LL_UCPD_IsEnableIT_TxUND
	TYPECEVT1IE	LL_UCPD_DisableIT_TypeCEventCC1
		LL_UCPD_EnableIT_TypeCEventCC1
	TYPECEVT2IE	LL_UCPD_DisableIT_TypeCEventCC2
		LL_UCPD_EnableIT_TypeCEventCC2
IMR2	TYPECEVT1IE	LL_UCPD_IsEnableIT_TypeCEventCC1
RX_ORDEXT1	RXSOPX1	LL_UCPD_SetRxOrdExt1
RX_ORDEXT2	RXSOPX2	LL_UCPD_SetRxOrdExt2
RX_ORDSET	RXORDSET	LL_UCPD_ReadRxOrderSet
RX_PAYSZ	RXPAYSZ	LL_UCPD_ReadRxPaySize
SR	FRSEVT	LL_UCPD_IsActiveFlag_FRS
	HRSTDISC	LL_UCPD_IsActiveFlag_TxHRSTDISC
	HRSTSENT	LL_UCPD_IsActiveFlag_TxHRSTSENT
	RXHRSTDET	LL_UCPD_IsActiveFlag_RxHRST
	RXMSGEND	LL_UCPD_IsActiveFlag_RxMsgEnd
	RXNE	LL_UCPD_IsActiveFlag_RxNE
	RXORDDET	LL_UCPD_IsActiveFlag_RxOrderSet
	RXOVR	LL_UCPD_IsActiveFlag_RxOvr
	TXIS	LL_UCPD_GetTypeCVstateCC1
		LL_UCPD_GetTypeCVstateCC2
		LL_UCPD_IsActiveFlag_TxIS
	TXMSGABT	LL_UCPD_IsActiveFlag_TxMSGABT
	TXMSGDISC	LL_UCPD_IsActiveFlag_TxMSGDISC
	TXMSGSENT	LL_UCPD_IsActiveFlag_TxMSGSENT
	TXUND	LL_UCPD_IsActiveFlag_TxUND
	TYPECEVT1	LL_UCPD_IsActiveFlag_TypeCEventCC1
	TYPECEVT2	LL_UCPD_IsActiveFlag_TypeCEventCC2
TXDR	DR	LL_UCPD_WriteData
	RXDATA	LL_UCPD_ReadData
TX_ORDSET	TXORDSET	LL_UCPD_WriteTxOrderSet
TX_PAYSZ	TXPAYSZ	LL_UCPD_WriteTxPaySize

124.31 USART

Table 55. Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
BRR	BRR	LL_USART_GetBaudRate
		LL_USART_SetBaudRate
CR1	CMIE	LL_USART_DisableIT_CM
		LL_USART_EnableIT_CM
		LL_USART_IsEnabledIT_CM
	DEAT	LL_USART_GetDEAssertionTime
		LL_USART_SetDEAssertionTime
	DEDT	LL_USART_GetDEDeassertionTime
		LL_USART_SetDEDeassertionTime
	EOBIE	LL_USART_DisableIT_EOB
		LL_USART_EnableIT_EOB
		LL_USART_IsEnabledIT_EOB
	FIFOEN	LL_USART_DisableFIFO
		LL_USART_EnableFIFO
		LL_USART_IsEnabledFIFO
	IDLEIE	LL_USART_DisableIT_IDLE
		LL_USART_EnableIT_IDLE
		LL_USART_IsEnabledIT_IDLE
	M0	LL_USART_ConfigCharacter
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	M1	LL_USART_ConfigCharacter
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	MME	LL_USART_DisableMuteMode
		LL_USART_EnableMuteMode
		LL_USART_IsEnabledMuteMode
	OVER8	LL_USART_GetOverSampling
		LL_USART_SetOverSampling
	PCE	LL_USART_ConfigCharacter
		LL_USART_GetParity
		LL_USART_SetParity
	PEIE	LL_USART_DisableIT_PE
		LL_USART_EnableIT_PE
		LL_USART_IsEnabledIT_PE
	PS	LL_USART_ConfigCharacter
		LL_USART_GetParity
		LL_USART_SetParity

Register	Field	Function
CR1	RE	LL_USART_DisableDirectionRx
		LL_USART_EnableDirectionRx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	RTOIE	LL_USART_DisableIT_RTO
		LL_USART_EnableIT_RTO
		LL_USART_IsEnabledIT_RTO
	RXFFIE	LL_USART_DisableIT_RXFF
		LL_USART_EnableIT_RXFF
		LL_USART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	LL_USART_DisableIT_RXNE_RXFNE
		LL_USART_EnableIT_RXNE_RXFNE
		LL_USART_IsEnabledIT_RXNE_RXFNE
	TCIE	LL_USART_DisableIT_TC
		LL_USART_EnableIT_TC
		LL_USART_IsEnabledIT_TC
	TE	LL_USART_DisableDirectionTx
		LL_USART_EnableDirectionTx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	TXEIE_TXFNFIE	LL_USART_DisableIT_TXE_TXFNF
		LL_USART_EnableIT_TXE_TXFNF
		LL_USART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_USART_DisableIT_TXFE
		LL_USART_EnableIT_TXFE
		LL_USART_IsEnabledIT_TXFE
	UE	LL_USART_Disable
		LL_USART_Enable
		LL_USART_IsEnabled
	UESM	LL_USART_DisableInStopMode
		LL_USART_EnableInStopMode
		LL_USART_IsEnabledInStopMode
	WAKE	LL_USART_GetWakeUpMethod
		LL_USART_SetWakeUpMethod
CR2	ABREN	LL_USART_DisableAutoBaudRate
		LL_USART_EnableAutoBaudRate
		LL_USART_IsEnabledAutoBaud
	ABRMODE	LL_USART_GetAutoBaudRateMode
		LL_USART_SetAutoBaudRateMode
	ADD	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddress

Register	Field	Function
CR2	ADDM7	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddressLen
	CLKEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSCLKOutput
		LL_USART_EnableSCLKOutput
		LL_USART_IsEnabledSCLKOutput
	CPHA	LL_USART_ConfigClock
		LL_USART_GetClockPhase
		LL_USART_SetClockPhase
	CPOL	LL_USART_ConfigClock
		LL_USART_GetClockPolarity
		LL_USART_SetClockPolarity
	DATAINV	LL_USART_GetBinaryDataLogic
		LL_USART_SetBinaryDataLogic
	DIS_NSS	LL_USART_DisableSPISlaveSelect
		LL_USART_EnableSPISlaveSelect
		LL_USART_IsEnabledSPISlaveSelect
	LBCL	LL_USART_ConfigClock
		LL_USART_GetLastClkPulseOutput
		LL_USART_SetLastClkPulseOutput
	LBDIE	LL_USART_DisableIT_LBD
		LL_USART_EnableIT_LBD
		LL_USART_IsEnabledIT_LBD
	LBDL	LL_USART_GetLINBrkDetectionLen
		LL_USART_SetLINBrkDetectionLen
	LINEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableLIN
		LL_USART_EnableLIN
		LL_USART_IsEnabledLIN

Register	Field	Function
CR2	MSBFIRST	LL_USART_GetTransferBitOrder
		LL_USART_SetTransferBitOrder
	RTOEN	LL_USART_DisableRxTimeout
		LL_USART_EnableRxTimeout
		LL_USART_IsEnabledRxTimeout
	RXINV	LL_USART_GetRXPinLevel
		LL_USART_SetRXPinLevel
	SLVEN	LL_USART_DisableSPISlave
		LL_USART_EnableSPISlave
		LL_USART_IsEnabledSPISlave
	STOP	LL_USART_ConfigCharacter
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigSmartcardMode
		LL_USART_GetStopBitsLength
		LL_USART_SetStopBitsLength
	SWAP	LL_USART_GetTXRXSwap
		LL_USART_SetTXRXSwap
	TXINV	LL_USART_GetTXPinLevel
		LL_USART_SetTXPinLevel
CR3	CTSE	LL_USART_DisableCTSHWFlowCtrl
		LL_USART_EnableCTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	CTSIE	LL_USART_DisableIT_CTS
		LL_USART_EnableIT_CTS
		LL_USART_IsEnabledIT_CTS
	DDRE	LL_USART_DisableDMADeactOnRxErr
		LL_USART_EnableDMADeactOnRxErr
		LL_USART_IsEnabledDMADeactOnRxErr
	DEM	LL_USART_DisableDEMode
		LL_USART_EnableDEMode
		LL_USART_IsEnabledDEMode
	DEP	LL_USART_GetDESignalPolarity
		LL_USART_SetDESignalPolarity
	DMAR	LL_USART_DisableDMAReq_RX
		LL_USART_EnableDMAReq_RX
		LL_USART_IsEnabledDMAReq_RX
	DMAT	LL_USART_DisableDMAReq_TX
		LL_USART_EnableDMAReq_TX
		LL_USART_IsEnabledDMAReq_TX

Register	Field	Function
CR3	EIE	LL_USART_DisableIT_ERROR
		LL_USART_EnableIT_ERROR
		LL_USART_IsEnabledIT_ERROR
	HDSEL	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableHalfDuplex
		LL_USART_EnableHalfDuplex
		LL_USART_IsEnabledHalfDuplex
	IREN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableIrda
		LL_USART_EnableIrda
		LL_USART_IsEnabledIrda
	IRLP	LL_USART_GetIrdaPowerMode
		LL_USART_SetIrdaPowerMode
	NACK	LL_USART_DisableSmartcardNACK
		LL_USART_EnableSmartcardNACK
		LL_USART_IsEnabledSmartcardNACK
	ONEBIT	LL_USART_DisableOneBitSamp
		LL_USART_EnableOneBitSamp
		LL_USART_IsEnabledOneBitSamp
	OVRDIS	LL_USART_DisableOverrunDetect
		LL_USART_EnableOverrunDetect
		LL_USART_IsEnabledOverrunDetect
	RTSE	LL_USART_DisableRTSHWFlowCtrl
		LL_USART_EnableRTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	RXFTCFG	LL_USART_ConfigFIFOsThreshold
		LL_USART_GetRXFIFOThreshold
		LL_USART_SetRXFIFOThreshold
	RXFTIE	LL_USART_DisableIT_RXFT

Register	Field	Function
CR3	RXFTIE	LL_USART_EnableIT_RXFT
		LL_USART_IsEnabledIT_RXFT
	SCARCNT	LL_USART_GetSmartcardAutoRetryCount
		LL_USART_SetSmartcardAutoRetryCount
	SCEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSmartcard
		LL_USART_EnableSmartcard
		LL_USART_IsEnabledSmartcard
	TCBGTIE	LL_USART_DisableIT_TCBGT
		LL_USART_EnableIT_TCBGT
		LL_USART_IsEnabledIT_TCBGT
	TXFTCFG	LL_USART_ConfigFIFOsThreshold
		LL_USART_GetTXFIFOThreshold
		LL_USART_SetTXFIFOThreshold
	TXFTIE	LL_USART_DisableIT_TXFT
		LL_USART_EnableIT_TXFT
		LL_USART_IsEnabledIT_TXFT
GTPR	GT	LL_USART_GetSmartcardGuardTime
		LL_USART_SetSmartcardGuardTime
	PSC	LL_USART_GetIrdaPrescaler
		LL_USART_GetSmartcardPrescaler
		LL_USART_SetIrdaPrescaler
		LL_USART_SetSmartcardPrescaler
ICR	CMCF	LL_USART_ClearFlag_CM
	CTSCF	LL_USART_ClearFlag_nCTS
	EOBCF	LL_USART_ClearFlag_EOB
	FECF	LL_USART_ClearFlag_FE
	IDLECF	LL_USART_ClearFlag_IDLE
	LBDCF	LL_USART_ClearFlag_LBD
	NECF	LL_USART_ClearFlag_NE
	ORECF	LL_USART_ClearFlag_ORE
	PECF	LL_USART_ClearFlag_PE
	RTOCF	LL_USART_ClearFlag_RTO
	TCBGTCF	LL_USART_ClearFlag_TCBGT
	TCCF	LL_USART_ClearFlag_TC

Register	Field	Function
ICR	TXFECF	LL_USART_ClearFlag_TXFE
	UDRCF	LL_USART_ClearFlag_UDR
ISR	ABRE	LL_USART_IsActiveFlag_ABRE
	ABRF	LL_USART_IsActiveFlag_ABR
	BUSY	LL_USART_IsActiveFlag_BUSY
	CMF	LL_USART_IsActiveFlag_CM
	CTS	LL_USART_IsActiveFlag_CTS
	CTSIF	LL_USART_IsActiveFlag_nCTS
	EOBF	LL_USART_IsActiveFlag_EOB
	FE	LL_USART_IsActiveFlag_FE
	IDLE	LL_USART_IsActiveFlag_IDLE
	LBDF	LL_USART_IsActiveFlag_LBD
	NE	LL_USART_IsActiveFlag_NE
	ORE	LL_USART_IsActiveFlag_ORE
	PE	LL_USART_IsActiveFlag_PE
	REACK	LL_USART_IsActiveFlag_REACK
	RTOF	LL_USART_IsActiveFlag_RTO
	RWU	LL_USART_IsActiveFlag_RWU
	RXFF	LL_USART_IsActiveFlag_RXFF
	RXFT	LL_USART_IsActiveFlag_RXFT
	RXNE_RXFNE	LL_USART_IsActiveFlag_RXNE_RXFNE
	SBKF	LL_USART_IsActiveFlag_SBK
	TC	LL_USART_IsActiveFlag_TC
	TCBGT	LL_USART_IsActiveFlag_TCBGT
	TEACK	LL_USART_IsActiveFlag_TEACK
	TXE_TXFNF	LL_USART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_USART_IsActiveFlag_TXFE
	TXFT	LL_USART_IsActiveFlag_TXFT
	UDR	LL_USART_IsActiveFlag_UDR
PRESC	PRESCALER	LL_USART_GetPrescaler
		LL_USART_SetPrescaler
RDR	RDR	LL_USART_DMA_GetRegAddr
		LL_USART_ReceiveData8
		LL_USART_ReceiveData9
RQR	ABRRQ	LL_USART_RequestAutoBaudRate
	MMRQ	LL_USART_RequestEnterMuteMode
	RXFRQ	LL_USART_RequestRxDataFlush
	SBKRQ	LL_USART_RequestBreakSending
	TXFRQ	LL_USART_RequestTxDataFlush
RTOR	BLEN	LL_USART_GetBlockLength
		LL_USART_SetBlockLength

Register	Field	Function
RTOR	RTO	LL_USART_GetRxTimeout
		LL_USART_SetRxTimeout
TDR	TDR	LL_USART_DMA_GetRegAddr
		LL_USART_TransmitData8
		LL_USART_TransmitData9

124.32 WWDG

Table 56. Correspondence between WWDG registers and WWDG low-layer driver functions

Register	Field	Function
CFR	EWI	LL_WWDG_EnableIT_EWKUP
		LL_WWDG_IsEnabledIT_EWKUP
	W	LL_WWDG_GetWindow
		LL_WWDG_SetWindow
	WDGTB	LL_WWDG_GetPrescaler
		LL_WWDG_SetPrescaler
CR	T	LL_WWDG_GetCounter
		LL_WWDG_SetCounter
	WDGA	LL_WWDG_Enable
		LL_WWDG_IsEnabled
SR	EWIF	LL_WWDG_ClearFlag_EWKUP
		LL_WWDG_IsActiveFlag_EWKUP

125 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
  - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
  - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 Series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not require in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, …). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32U5 devices. To ensure compatibility between all devices and portability with others Series and lines, the API is split into the generic and the extension APIs . For more details, please refer to *section* ***Devices supported by the HAL drivers***.

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32u5xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration…)

A template is provided in the HAL drivers folders (stm32u5xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32u5xx_hal.h file has to be included.

What is the difference between xx_hal_ppp.c/.h and xx_hal_ppp_ *ex* .c/.h?

The HAL driver architecture supports common features across STM32 Series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32u5xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32u5xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32u5xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef *pHandle* structure located in each driver in addition to the Initialization structure

PPP_HandleTypeDef *pHandle is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32u5xx_hal_msp.c. A template is provided in the HAL driver folders (stm32u5xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute *__weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,…), configure the systTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling ***HAL_RCC_ClockConfig()*** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling *HAL_IncTick()* function in SysTick ISR and retrieve the value of this variable by calling *HAL_GetTick()* function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,…).

2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,..) by calling HAL_PPP_MspInit() in stm32u5xx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32u5xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

When should I use HAL versus LL drivers?

HAL drivers offer high-level and function-oriented APIs, with a high level of portability. Product/IPs complexity is hidden for end users. LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of product/IPs specifications.

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. Source code shall directly include the necessary stm32u5xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with HAL and then manage the I/O operations with LL drivers. The major difference between HAL and LL is that HAL drivers require to create and use handles for operation management while LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in Examples_MIX example.

Is there any LL APIs which are not available with HAL?

Yes, there are. A few Cortex® APIs have been added in stm32u5xx_ll_cortex.h e.g. for accessing SCB or SysTick registers.

Why are SysTick interrupts not enabled on LL drivers?

When using LL drivers in standalone mode, you do not need to enable SysTick interrupts because they are not used in LL APIs, while HAL functions requires SysTick interrupts to manage timeouts.

Revision history

Table 57. Document revision history

Date	Revision	Changes
28-Sep-2021	1	Initial release.
22-Feb-2023	2	Removed details on the status bit and flag read from registers from Section 3.5.2: HAL general naming rules. Updated Section 3.12.4.1: Timeout management Document updated to cover all HAL and low-lower drivers for the STM32U5 series: • STM32U535/545 • STM32U575/585 revision W • STM32U59x/5Ax revision X
07-Aug-2023	3	Updated document to cover all HAL and low-lower drivers for the following STM32U5 part numbers: • STM32U535/545 revision Z • STM32U575/585 revision W • STM32U59x/5Ax revision X • STM32U5Fx/5Gx revision Z
20-Mar-2024	4	Removed section *Updating existing common APIs* from Section 3.7.2: HAL extension model cases. Updated document to cover all HAL and low-lower drivers for the following STM32U5 part numbers: • STM32U535/545 revision Y • STM32U575/585 revision W • STM32U59x/5Ax revision W • STM32U5Fx/5Gx revision Z

Contents

List of tables

List of figures