Description of STM32C0 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 STM32CubeC0 for STM32C0 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, FileX, and LevelX.

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 peripheral-oriented. For example, the timer APIs are split into several categories following the peripheral functions, such as basic timer, capture, and pulse width modulation (PWM). The HAL driver layer implements runtime failure detection by checking the input values of all functions. Such dynamic checking enhances the firmware robustness. Runtime 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.

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 CodeSonar® 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 STM32CubeC0 MCU Package runs on STM32C0 32-bit microcontrollers based on the Arm® Cortex®-M processor.

Note: Arm is a registered trademark 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
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CMSIS	Cortex microcontroller software interface standard
CPU	Central processing unit
CRC	CRC calculation unit
CSS	Clock security system
DLYB	Delay block
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General-purpose I/Os
HAL	Hardware abstraction layer
HCD	USB host controller driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
ICACHE	Instruction cache
IRDA	Infrared data association
IWDG	Independent watchdog
MCO	Microcontroller clock output
MPU	Memory protection unit
MSP	MCU specific package
NVIC	Nested vectored interrupt controller
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PWR	Power controller
RCC	Reset and clock controller
RTC	Real-time clock
SMBUS	System management bus
SPI	Serial peripheral interface
SRAM	SRAM external memory
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
USB	Universal serial bus

Acronym	Definition
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 a 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
stm32c0xx_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32c0xx_hal_adc.c, stm32c0xx_hal_irda.c.*
stm32c0xx_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:stm32c0xx_hal_adc.h,stm32c0xx_hal_irda.h.*
stm32c0xx_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 a different way. *Example:stm32c0xx_hal_adc_ex.c,stm32c0xx_hal_flash_ex.c.*
stm32c0xx_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: stm32c0xx_hal_adc_ex.h,stm32c0xx_hal_flash_ex.h.*
stm32c0xx_hal.c	This file is used for HAL initialization and contains DBGMCU, remap and time delay based on SysTick APIs.
stm32c0xx_hal.h	stm32c0xx_hal.c header file
stm32c0xx_hal_msp_template.c	Template file to be copied to the user application folder It contains the MSP initialization and deinitialization (main routine and callbacks) of the peripheral used in the user application.
stm32c0xx_hal_conf_template.h	Template file allowing the drivers to be customized for a given application
stm32c0xx_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_stm32c0xx.c	This file contains SystemInit() that is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows the relocation of the vector table in internal SRAM.
startup_stm32c0xx.s	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows the adaptation of the stack/heap size to fit the application requirements.
stm32c0xx_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly the adaptation of the stack/heap size to fit the application requirements.
stm32c0xx_hal_msp.c	This file contains the MSP initialization and deinitialization (main routine and callbacks) of the peripheral used in the user application.

File	Description
stm32c0xx_hal_conf.h	This file allows the HAL drivers to be customized for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32c0xx_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 *stm32c0xx*_hal.c) used as HAL timebase. By default, this function is called each 1 ms 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.

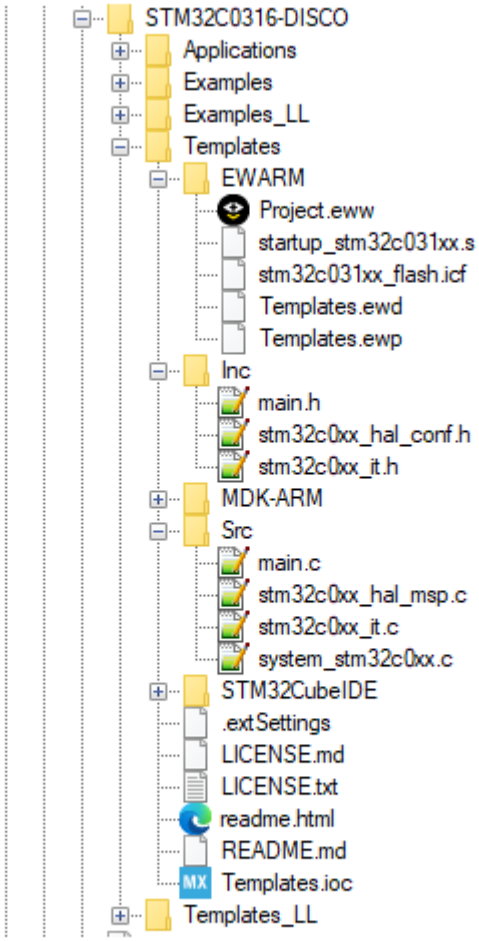
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

Each project template provides an 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 configures 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 peripheral 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 also used 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) nonconstant 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 UART communication baudrate.*/
uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received in a fram
e.*/
uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
uint32_t Parity; /*!< Specifies the parity mode. */
uint32_t Mode; /*!< Specifies wether the Receive or Transmit mode is enabled or disabled.*/
uint32_t HwFlowCtl; /*!< Specifies wether the hardware flow control mode is enabled or disabl
ed.*/
uint32_t OverSampling; /*!< Specifies wether the Over sampling 8 is enabled or disabled,
to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```

Note: The config structure is used to initialize the submodules or subinstances. 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 subcategories :
  - **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 Sin
gleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDi
ff);
```

  - **Feature/Device specific APIs**: these APIs are implemented in the extension file and delimited by specific define statements relative to CMSIS bits or features and depending on device 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

Peripheral/module	STM32C092xx	STM32C051xx	STM32C071xx	STM32C031xx	STM32C011xx
stm32c0xx_hal.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_adc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_adc_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_cortex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_crc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_crc_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_dma.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_dma_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_exti.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_fdcan.c	Yes	No	No	No	No
stm32c0xx_hal_flash.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_flash_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_fdcan.c	Yes	No	No	No	No
stm32c0xx_hal_gpio.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_hcd.c	No	No	Yes	No	No
stm32c0xx_hal_i2c.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_i2s.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_i2s_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_irda.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_iwdg.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_msp_template.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_pcd.c	No	No	Yes	No	No
stm32c0xx_hal_pcd_ex.c	No	No	Yes	No	No
stm32c0xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_pwr_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_rcc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_rcc_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_rtc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_rtc_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_smartcard.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_smartcard_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_smbus.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_spi.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_tim.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_tim_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_uart.c	Yes	Yes	Yes	Yes	Yes

Peripheral/module	STM32C092xx	STM32C051xx	STM32C071xx	STM32C031xx	STM32C011xx
stm32c0xx_hal_uart_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_usart.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_usart_ex.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_hal_wwdg.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_adc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_crc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_crs.c	No	No	Yes	No	No
stm32c0xx_ll_dma.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_exti.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_gpio.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_i2c.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_pwr.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_rcc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_rtc.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_spi.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_tim.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_usart.c	Yes	Yes	Yes	Yes	Yes
stm32c0xx_ll_usb.c	No	No	Yes	No	No
stm32c0xx_ll_utils.c	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	*stm32c0xx_hal_ppp (c/h)*	*stm32c0xx_hal_ppp_ex (c/h)*	*stm32c0xx_ 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 STM32C0 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in stm32c0xxx.h header file:
stm32c0xxx.h corresponds to stm32c031xx.h and stm32c011xx.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 the specified interrupt

- NVIC and stm32c0xx_hal_cortex.c are two Arm® Cortex® core features. The APIs related to these features are located in the stm32c0xx_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))
```

  - Pseudocode 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 stm32c0xx_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 the peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Example: HAL_USART_ErrorCallback Called by the 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 deinitialization functions:**HAL_PPP_Init(), HAL_PPP_DeInit()
- **I/O operation functions**: HAL_PPP_Read(), HAL_PPP_Write(),HAL_PPP_Transmit(), HAL_PPP_Receive()
- **Control functions**: HAL_PPP_Set (), HAL_PPP_Get ().
- **State and error 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 deinitialization functions initialize a given peripheral and configure 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 I/O operation functions perform a row access to the peripheral payload data in write and read modes.

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

The peripheral state and errors functions retrieve in runtime 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 (such as 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.
I/O 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 waits for the end of conversions when the polling method is used. In this case, a timeout 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 been 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 gets in runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction gets in runtime the error that occurred during an 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, stm32c0xx_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32c0xx_hal_ppp_ex.h) for a given part number.

Below is 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_Calibration_GetValue()	This function is used to get the ADC calibration factor

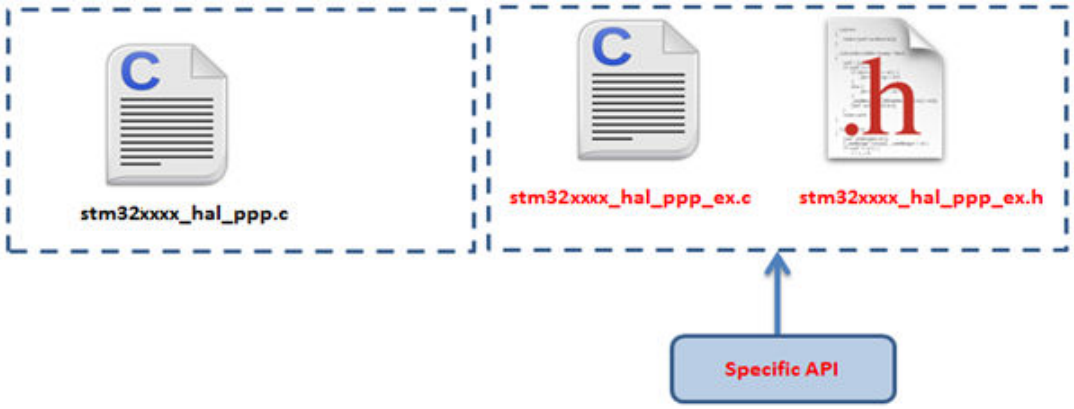
3.7.2 HAL extension model cases

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 stm32c0xx_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

Figure 2. Adding device-specific functions



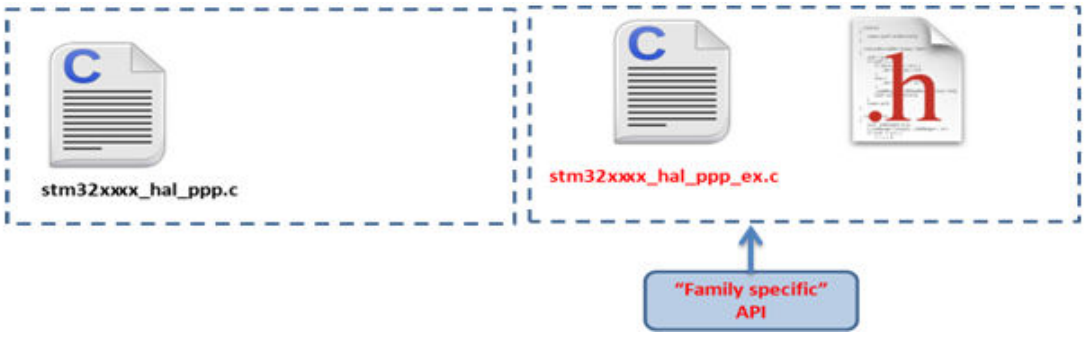
Example: stm32c0xx_hal_pwr_ex.c/h

```
#if defined(PWR_SHDW_SUPPORT)
void HAL_PWREx_EnterSHUTDOWNMode(void);
#endif
```

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

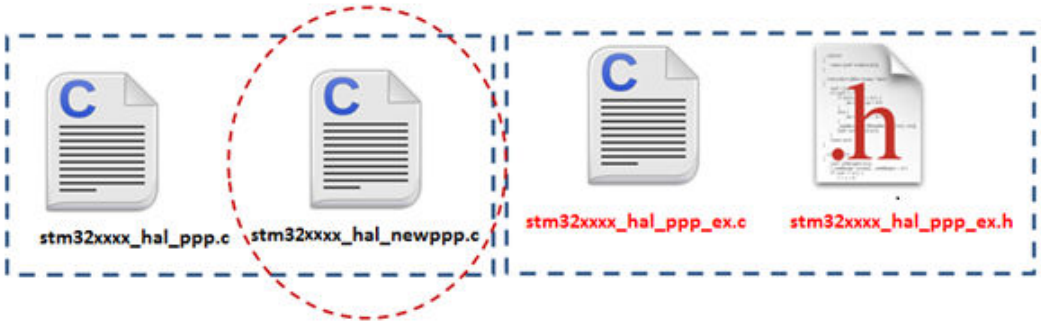


Adding a new peripheral (specific to a device belonging to a given family)

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 stm32c0xx_hal_newppp.c. However, the inclusion of this file is selected in the stm32c0xx_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. **Adding new peripherals**



Example: stm32c0xx_hal_adc.c/h

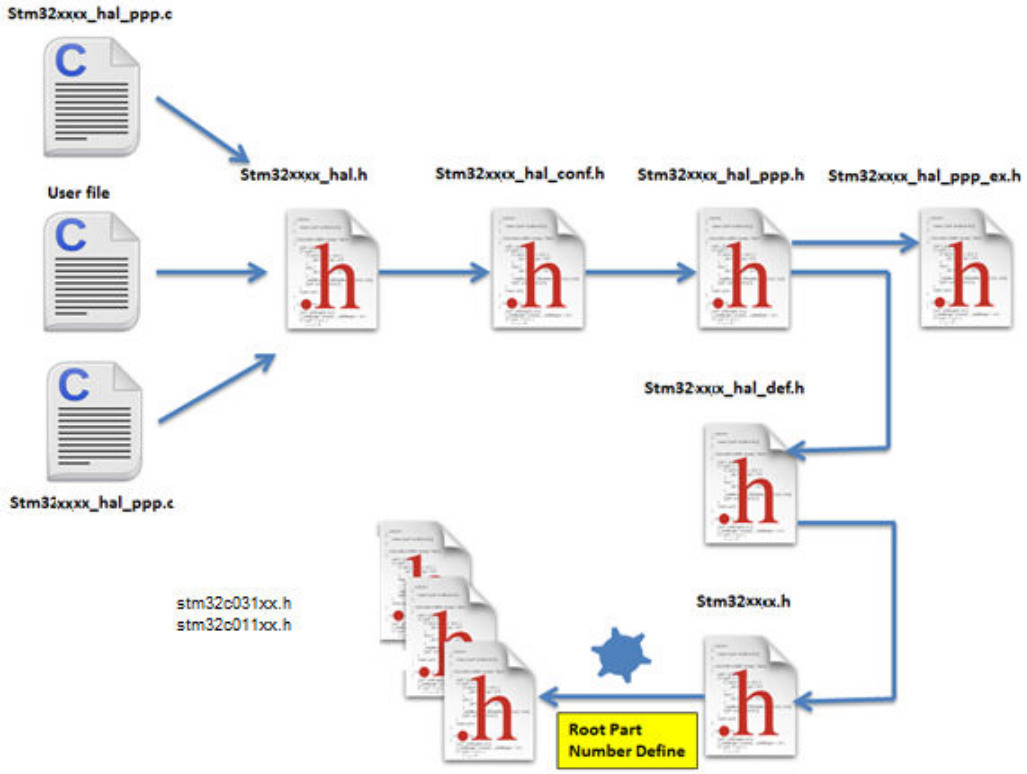
Updating existing data structures

The data structure for a specific device part number (for example PPP_InitTypeDef) can have different fields. In this case, the data structure is defined in the extension header file and delimited by the specific part number define statement.

3.8 File inclusion model

The header of the common HAL driver file (stm32c0xx_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 stm32c0xx_hal_conf.h
* @author MCD Application Team
* @version VX.Y.Z * @date dd-mm-yyyy
* @brief This file contains the modules to be used
******************************************************************************
(…)
#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 *stm32c0xx_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 the 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 stm32c0xx_hal_def.h file calls the stm32c0xx.h file in the 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, stm32c0xx_hal_conf.h, performs the customization of 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	Value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	48 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE startup, expressed in ms	100 ms
HSI_VALUE	Value of the internal oscillator (HSI) expressed in Hz.	48 000 000 Hz
HSI48_VALUE	Value of the internal high-speed oscillator for USB expressed in Hz. The real value may vary depending on the variations in voltage and temperature.	48 000 000 Hz
LSI_VALUE	Default value of the low-speed internal oscillator (LSI) expressed in Hz.	32000 Hz
LSE_VALUE	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 startup, expressed in ms	5000
VDD_VALUE	V_{DD} value	3300 mV
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	FALSE
INSTRUCTION_CACHE_ENABLE	Enables ICACHE feature	TRUE

Note: The stm32c0xx_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 stm32c0xx_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, LSE, LSI).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - Selects the system clock source
  - Configures AHB and APB1 clock dividers
  - Configures the number of flash memory wait states
  - Updates the SysTick configuration when the HCLK clock changes.

Some peripheral clocks are not derived from the system clock (such as RTC). In this case, the clock configuration is performed by an extended API defined in stm32c0xx_hal_rcc_ex.c:
*HAL_RCCEx_PeriphCLKConfig(RCC_PeriphCLKInitTypeDef *PeriphClkInit)*.

Additional RCC HAL driver functions are available:

- HAL_RCC_DeInit() clock deinitialization function that returns clock configuration to the reset state
- Get clock functions that retrieve the various clock configurations (such as system clock, HCLK or PCLK1)
- MCO and CSS configuration functions

A set of macros are defined in stm32c0xx_hal_rcc.h and stm32c0xx_hal_rcc_ex.h. They enable the execution of elementary operations on RCC block registers, such as peripheral 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 ().

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 stm32c0xx_it.c and implement HAL_GPIO_EXTI_Rising_Callback() and/or HAL_GPIO_EXTI_Falling_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

Structure field	Description
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 an 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, stm32c0xx_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()

3.11.4 PWR

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

- Wake-up pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTANDBYMode()
- STM32C0 low-power configuration features:
  - HAL_PWREx_EnterSHUTDOWNMode()

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 configures a given I/O as an external interrupt or external event.

The EXTI lines connected internally to the RTC, I2C1, and USART1, are configured within the HAL drivers of these peripherals 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()
__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 stm32c0xx_it.c file, and implement HAL_PPP_FUNCTIONCallback() callback function (for example HAL_PWR_PVDCallback().

3.11.6 DMA

The DMA HAL driver enables and configures the peripheral to be connected to the DMA channels (except for internal SRAM/flash memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given channel, HAL_DMA_Init() API programs the required configuration through the following parameters:

- Transfer direction
- Source and destination data formats
- Circular, normal control mode
- Channel priority level
- Source and destination increment mode
- Hardware request connected to the peripheral

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

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 I/O operation functions").

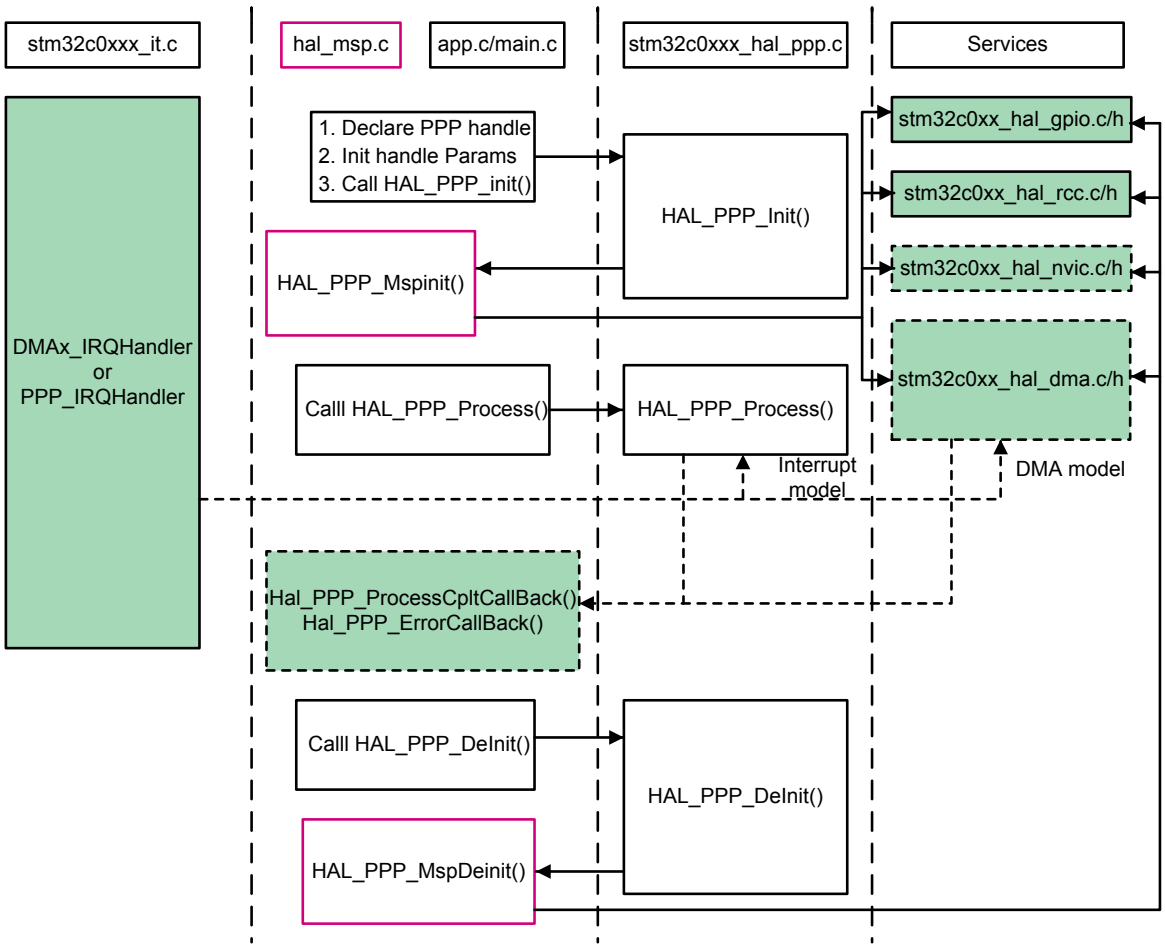
Note: DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However, when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

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. Nondotted 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 deinitialization functions, a set of APIs are provided to initialize the HAL core implemented in file stm32c0xx_hal.c.

- HAL_Init(): this function must be called at application startup to
  - Initialize data/instruction cache and prefetch queue
  - Set SysTick timer to generate an interrupt each 1 ms (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 a “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 deinitializations.

- HAL_GetTick(): this function gets the current SysTick counter value (incremented in SysTick interrupt) used by peripheral 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 the 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 48 MHz based on the HSE clock.

```
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

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 the 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 deinitialization:

```
/**
* @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 their own initialization routine.

The HAL MSP callback is implemented inside the *stm32c0xx_hal_msp.c* file in the user folders. An *stm32c0xx_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 the 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.

stm32c0xx_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 deinitialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP deinitialization routine

By default, if no peripheral needs to be deinitialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP deinitialization 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 deinitialized in runtime 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 *stm32c0xx_it.c.*

The *HAL_PPP_ProcessCpltCallback()* function is declared as a 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)
{
}
```

stm32c0xx_it.c file:

```
extern UART_HandleTypeDef UartHandle;
void USART1_IRQHandler(void)
{
HAL_UART_IRQHandler(&UartHandle);
}
```

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 *stm32c0xx_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 with 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 a weak function in the driver that means. The user can declare it again in the application code. The function in the driver must not be modified.

An example of use is illustrated below:

main.c file:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
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_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

stm32c0xx_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 until 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 until the process is successful

1. *HAL_MAX_DELAY is defined in the stm32c0xx_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 retrieve 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 *Runtime checking*

The HAL implements runtime failure detection by checking the input values of all HAL driver functions. The runtime 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 verifies that the input value lies within the parameter allowed values.

To enable the runtime checking, use the *assert_param* macro, and leave the define ***USE_FULL_ASSERT*** uncommented in the *stm32c0xx_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 stm32c0xx_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 runtime 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 deinitialization (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.

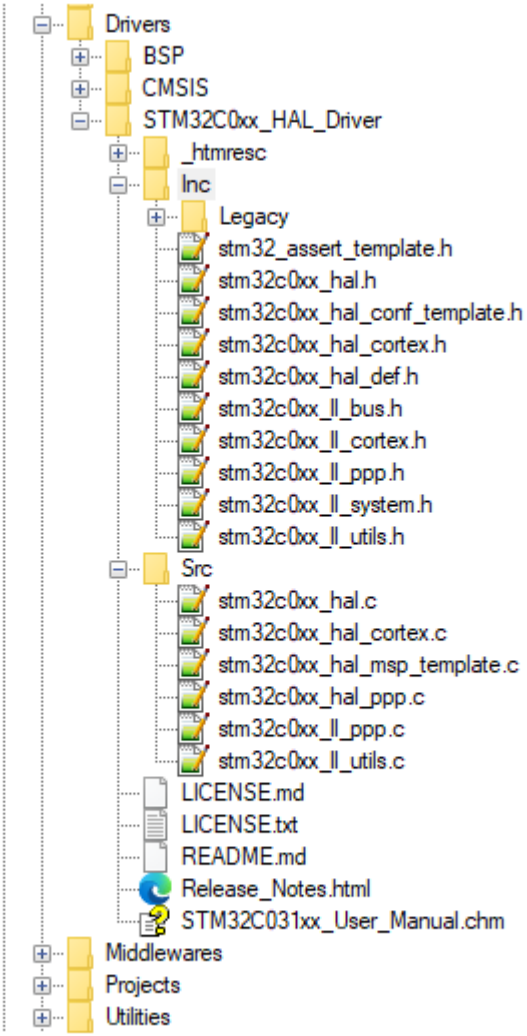
Table 16. LL driver files

File	Description
stm32c0xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB1_GRP1_EnableClock*
stm32c0xx_ll_ppp.h/.c	stm32c0xx_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 the stm32c0xx_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32c0xx_ll_ppp.h file.
stm32c0xx_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")
stm32c0xx_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.
stm32c0xx_ll_system.h	System-related operations. Example: LL_SYSCFG_xxx, LL_DBGMCU_xxx and LL_FLASH_xxx
stm32_assert_template.h	Template file defines the assert_param macro that is used when runtime 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.

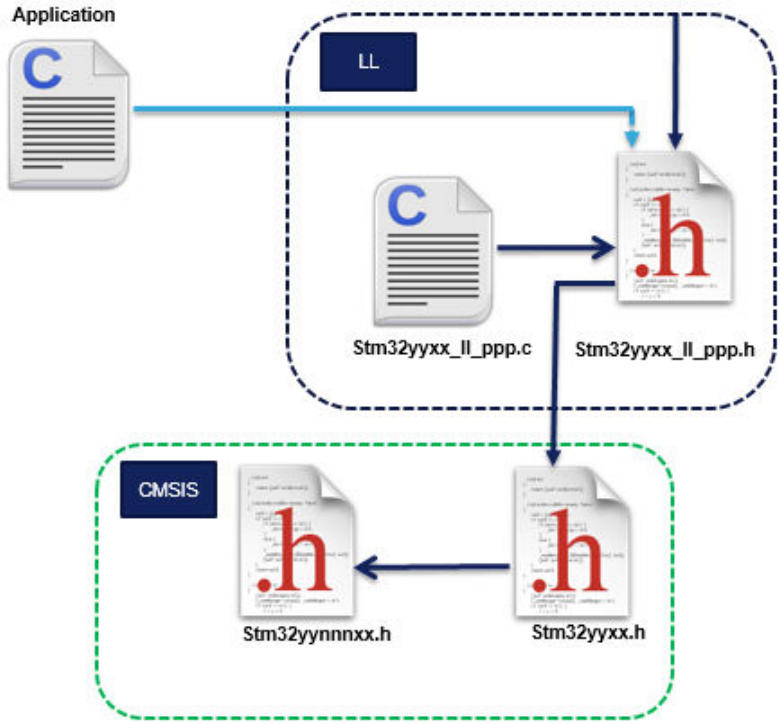
Figure 7. Low-layer driver folders



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 the stm32c0xx_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 deinitialization (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*	Deinitializes 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)
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 *Runtime checking*

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

When using the LL drivers in standalone mode (without calling HAL functions), the following actions are required to use runtime 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 runtime checking is enabled.
2. Include the 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: Runtime 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 on 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_AHB1_GRP1_EnableClock (LL_AHB1_GRP1_PERIPH_DMA1) • LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_RTC)

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 setting

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 stm32c0xx_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 STM32CubeC0 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 the stm32c0xx 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 HAL Initialization and Configuration functions

This section provides functions allowing to:

- Initialize the Flash interface the NVIC allocation and initial time base clock configuration.
- De-initialize 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

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 STOP mode
- 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:

- Enable/Disable Pin remap

This section contains the following APIs:

- *HAL_SYSCFG_EnableRemap()*
- *HAL_SYSCFG_DisableRemap()*
- *HAL_SYSCFG_SetPinBinding()*
- *HAL_SYSCFG_GetPinBinding()*

6.1.6 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

Configure the Flash prefetch and the Instruction cache, 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 stm32c0xx_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 HSI clock, as HSI is the clock used after a system Reset. 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 source of time base.

Return values

- **HAL:** status

Notes

- This function is optional.

HAL_MspInit

Function name

void HAL_MspInit (void)

Function description

Initialize 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: The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.

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

Returns 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

Returns 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

Returns 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 STOP mode.

Return values

- **None:**

HAL_DBGMCU_DisableDBGStopMode

Function name

void HAL_DBGMCU_DisableDBGStopMode (void)

Function description

Disable the Debug Module during STOP mode.

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_EnableRemap

Function name

void HAL_SYSCFG_EnableRemap (uint32_t PinRemap)

Function description

Enable the remap on PA11_PA12.

Parameters

- **PinRemap:** specifies which pins have to be remapped This parameter can be any combination of the following values:
  - SYSCFG_REMAP_PA11
  - SYSCFG_REMAP_PA12

Return values

- **None:**

HAL_SYSCFG_DisableRemap

Function name

void HAL_SYSCFG_DisableRemap (uint32_t PinRemap)

Function description

Disable the remap on PA11_PA12.

Parameters

- **PinRemap:** specifies which pins will behave normally This parameter can be any combination of the following values:
  - SYSCFG_REMAP_PA11
  - SYSCFG_REMAP_PA12

Return values

- **None:**

HAL_SYSCFG_SetPinBinding

Function name

void HAL_SYSCFG_SetPinBinding (uint32_t pin_binding)

Function description

Set Pin Binding.

Parameters

- **pin_binding:** specifies which pin will bind a specific GPIO for each die package This parameter can be a value of Bind Pin config

Return values

- **None:**

HAL_SYSCFG_GetPinBinding

Function name

uint32_t HAL_SYSCFG_GetPinBinding (uint32_t pin_binding_source)

Function description

return Pin Binding configuration

Parameters

- **pin_binding_source:** This parameter can be a value of Bind Pin Source

Return values

- **PinMux:** configuration

6.2 HAL Firmware driver defines

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

6.2.1 HAL

HAL

Bind Pin config

HAL_BIND_TSSOP20_PIN19_PA14

STM32C091/92 TSSOP20 package, Pin19 assigned to GPIO PA14

HAL_BIND_TSSOP20_PIN19_PA15

STM32C091/92 TSSOP20 package, Pin19 assigned to GPIO PA15

HAL_BIND_TSSOP20_PIN15_PA8

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PA8

HAL_BIND_TSSOP20_PIN15_PB0

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PB0

HAL_BIND_TSSOP20_PIN15_PB1

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PB1

HAL_BIND_TSSOP20_PIN15_PB2

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PB2

HAL_BIND_TSSOP20_PIN20_PB6

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB6

HAL_BIND_TSSOP20_PIN20_PB3

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB3

HAL_BIND_TSSOP20_PIN20_PB4

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB4

HAL_BIND_TSSOP20_PIN20_PB5

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB5

HAL_BIND_WLCSP24_PINA3_PB5

STM32C091/92 WLCSP24 package, PinA3 assigned to GPIO PB5

HAL_BIND_WLCSP24_PINA3_PB3

STM32C091/92 WLCSP24 package, PinA3 assigned to GPIO PB3

HAL_BIND_WLCSP24_PINB4_PB7

STM32C091/92 WLCSP24 package, PinB4 assigned to GPIO PB7

HAL_BIND_WLCSP24_PINB4_PB8

STM32C091/92 WLCSP24 package, PinB4 assigned to GPIO PB8

Bind Pin Source

HAL_BIND_TSSOP20_PIN19

STM32C051 WLCSP15 package, GPIO PinH3 multiplexer

HAL_BIND_TSSOP20_PIN15

STM32C051 WLCSP15 package, GPIO PinG2 multiplexer

HAL_BIND_TSSOP20_PIN20

STM32C051 WLCSP15 package, GPIO PinK3 multiplexer

HAL_BIND_WLCSP24_PINA3

STM32C051 WLCSP15 package, GPIO PinJ2 multiplexer

HAL_BIND_WLCSP24_PINB4

STM32C051 WLCSP15 package, GPIO PinH1 multiplexer

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

Get element from empty FIFO

HAL_FDCAN_ERROR_FIFO_FULL

Put element in full 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

IR Modulation Envelope signal selection

HAL_SYSCFG_IRDA_ENV_SEL_TIM16

00: Timer16 is selected as IR Modulation envelope source

HAL_SYSCFG_IRDA_ENV_SEL_USART1

01: USART1 is selected as IR Modulation envelope source

HAL_SYSCFG_IRDA_ENV_SEL_USART2

10: USART2 is selected as IR Modulation envelope source

IR output polarity selection

HAL_SYSCFG_IRDA_POLARITY_NOT_INVERTED

00: IR output polarity not inverted

HAL_SYSCFG_IRDA_POLARITY_INVERTED

01: IR output polarity inverted

HAL ISR Wrapper

HAL_SYSCFG_ITLINE0

Internal define for macro handling

HAL_SYSCFG_ITLINE2

Internal define for macro handling

HAL_SYSCFG_ITLINE3

Internal define for macro handling

HAL_SYSCFG_ITLINE4

Internal define for macro handling

HAL_SYSCFG_ITLINE5

Internal define for macro handling

HAL_SYSCFG_ITLINE6

Internal define for macro handling

HAL_SYSCFG_ITLINE7

Internal define for macro handling

HAL_SYSCFG_ITLINE9

Internal define for macro handling

HAL_SYSCFG_ITLINE10

Internal define for macro handling

HAL_SYSCFG_ITLINE11

Internal define for macro handling

HAL_SYSCFG_ITLINE12

Internal define for macro handling

HAL_SYSCFG_ITLINE13

Internal define for macro handling

HAL_SYSCFG_ITLINE14

Internal define for macro handling

HAL_SYSCFG_ITLINE15

Internal define for macro handling

HAL_SYSCFG_ITLINE16

Internal define for macro handling

HAL_SYSCFG_ITLINE19

Internal define for macro handling

HAL_SYSCFG_ITLINE20

Internal define for macro handling

HAL_SYSCFG_ITLINE21

Internal define for macro handling

HAL_SYSCFG_ITLINE22

Internal define for macro handling

HAL_SYSCFG_ITLINE23

Internal define for macro handling

HAL_SYSCFG_ITLINE24

Internal define for macro handling

HAL_SYSCFG_ITLINE25

Internal define for macro handling

HAL_SYSCFG_ITLINE26

Internal define for macro handling

HAL_SYSCFG_ITLINE27

Internal define for macro handling

HAL_SYSCFG_ITLINE28

Internal define for macro handling

HAL_SYSCFG_ITLINE29

Internal define for macro handling

HAL_SYSCFG_ITLINE30

Internal define for macro handling

HAL_SYSCFG_ITLINE31

Internal define for macro handling

HAL_ITLINE_WWDG

WWDG Interrupt

HAL_ITLINE_RTC

RTC Interrupt

HAL_ITLINE_FLASH_ITF

Flash ITF Interrupt

HAL_ITLINE_CLK_CTRL

CLK Control Interrupt

HAL_ITLINE_EXTI0

External Interrupt 0

HAL_ITLINE_EXTI1

External Interrupt 1

HAL_ITLINE_EXTI2

External Interrupt 2

HAL_ITLINE_EXTI3

External Interrupt 3

HAL_ITLINE_EXTI4

EXTI4 Interrupt

HAL_ITLINE_EXTI5

EXTI5 Interrupt

HAL_ITLINE_EXTI6

EXTI6 Interrupt

HAL_ITLINE_EXTI7

EXTI7 Interrupt

HAL_ITLINE_EXTI8

EXTI8 Interrupt

HAL_ITLINE_EXTI9

EXTI9 Interrupt

HAL_ITLINE_EXTI10

EXTI10 Interrupt

HAL_ITLINE_EXTI11

EXTI11 Interrupt

HAL_ITLINE_EXTI12

EXTI12 Interrupt

HAL_ITLINE_EXTI13

EXTI13 Interrupt

HAL_ITLINE_EXTI14

EXTI14 Interrupt

HAL_ITLINE_EXTI15

EXTI15 Interrupt

HAL_ITLINE_DMA1_CH1

DMA1 Channel 1 Interrupt

HAL_ITLINE_DMA1_CH2

DMA1 Channel 2 Interrupt

HAL_ITLINE_DMA1_CH3

DMA1 Channel 3 Interrupt

HAL_ITLINE_DMAMUX

DMAMUX Interrupt

HAL_ITLINE_DMA1_CH4

DMA1 Channel 4 Interrupt

HAL_ITLINE_DMA1_CH5

DMA1 Channel 5 Interrupt

HAL_ITLINE_DMA1_CH6

DMA1 Channel 6 Interrupt

HAL_ITLINE_DMA1_CH7

DMA1 Channel 7 Interrupt

HAL_ITLINE_ADC

ADC Interrupt

HAL_ITLINE_TIM1_BRK

TIM1 BRK Interrupt

HAL_ITLINE_TIM1_UPD

TIM1 UPD Interrupt

HAL_ITLINE_TIM1_TRG

TIM1 TRG Interrupt

HAL_ITLINE_TIM1_CCU

TIM1 CCU Interrupt

HAL_ITLINE_TIM1_CC

TIM1 CC Interrupt

HAL_ITLINE_TIM2

TIM2 Interrupt

HAL_ITLINE_TIM3

TIM3 Interrupt

HAL_ITLINE_TIM14

TIM14 Interrupt

HAL_ITLINE_TIM15

TIM15 Interrupt

HAL_ITLINE_TIM16

TIM16 Interrupt

HAL_ITLINE_TIM17

TIM17 Interrupt

HAL_ITLINE_I2C1

I2C1 Interrupt

HAL_ITLINE_I2C2

I2C1 Interrupt

HAL_ITLINE_SPI1

SPI1 Interrupt

HAL_ITLINE_SPI2

SPI1 Interrupt

HAL_ITLINE_USART1

USART1 GLB Interrupt

HAL_ITLINE_USART2

USART2 GLB Interrupt

HAL_ITLINE_USART3

USART3 GLB Interrupt

HAL_ITLINE_USART4

USART4 GLB Interrupt

HAL_ITLINE_FDCAN1_IT0

FDCAN1_IT0 GLB Interrupt

HAL_ITLINE_FDCAN1_IT1

FDCAN1_IT1 GLB Interrupt

Pin remapping

SYSCFG_REMAP_PA11

PA11 pad behaves digitally as PA9 GPIO pin

SYSCFG_REMAP_PA12

PA12 pad behaves digitally as PA10 GPIO pin

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

Boot Mode

SYSCFG_BOOT_MAINFLASH

Main Flash memory mapped at 0x0000 0000

SYSCFG_BOOT_SYSTEMFLASH

System Flash memory mapped at 0x0000 0000

SYSCFG_BOOT_SRAM

Embedded SRAM mapped at 0x0000 0000

Break

SYSCFG_BREAK_LOCKUP

Enables and locks the LOCKUP output of CortexM0+ with Break Input of TIM1/16/17

SYSCFG Exported Macros

__HAL_GET_PENDING_IT

Notes:

- Allow to determine interrupt source per line.

__HAL_SYSCFG_REMAPMEMORY_FLASH

__HAL_SYSCFG_REMAPMEMORY_SYSTEMFLASH

__HAL_SYSCFG_REMAPMEMORY_SRAM

__HAL_SYSCFG_GET_BOOT_MODE

Description:

- Return the boot mode as configured by user.

Return value:

- The: boot mode as configured by user. The returned value can be one of the following values

__HAL_SYSCFG_BREAK_LOCKUP_LOCK

Notes:

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

Enables and locks the connection of Cortex-M0+ LOCKUP (Hardfault) output to TIM1/16/17 Break input

__HAL_SYSCFG_FASTMODEPLUS_ENABLE

Description:

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

Parameters:

- __FASTMODEPLUS__: This parameter can be a value of

__HAL_SYSCFG_FASTMODEPLUS_DISABLE

__HAL_SYSCFG_IRDA_ENV_SELECTION

Description:

- selection of the modulation envelope signal macro, using bits [7:6] of SYSCFG_CFGR1 register

Parameters:

- __SOURCE__: This parameter can be a value of

__HAL_SYSCFG_GET_IRDA_ENV_SELECTION

__HAL_SYSCFG_IRDA_OUT_POLARITY_SELECTION

Description:

- IROut Polarity Selection, using bit[5] of SYSCFG_CFGR1 register.

Parameters:

- __SEL__: This parameter can be a value of

__HAL_SYSCFG_GET_POLARITY

Description:

- Return the IROut Polarity mode as configured by user.

Return value:

- The: IROut polarity as configured by user. The returned value can be one of

__HAL_SYSCFG_BREAK_ENABLE

Description:

- Break input to TIM1/16/17 capability enable/disable macros.

Parameters:

- __BREAK__: This parameter can be a value of

__HAL_SYSCFG_BREAK_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

SYSCFG_FASTMODEPLUS_PA9

Enable Fast mode Plus on PA9

SYSCFG_FASTMODEPLUS_PA10

Enable Fast mode Plus on PA10

SYSCFG_FASTMODEPLUS_PC14

Enable Fast mode Plus on PC14

Fast mode Plus driving capability activation for I2Cx

SYSCFG_FASTMODEPLUS_I2C1

Enable Fast mode Plus on I2C1

SYSCFG_FASTMODEPLUS_I2C2

Enable Fast mode Plus on I2C2

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_OversamplingTypeDef

ADC_OversamplingTypeDef is defined in the stm32c0xx_hal_adc.h

Data Fields

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

Field Documentation

- ***uint32_t ADC_OversamplingTypeDef::Ratio***
  Configures the oversampling ratio. 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***

7.1.2 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32c0xx_hal_adc.h

Data Fields

- ***uint32_t ClockPrescaler***
- ***uint32_t Resolution***
- ***uint32_t DataAlign***
- ***uint32_t ScanConvMode***
- ***uint32_t EOCSelection***
- ***FunctionalState LowPowerAutoWait***
- ***FunctionalState LowPowerAutoPowerOff***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t NbrOfConversion***
- ***FunctionalState DiscontinuousConvMode***
- ***uint32_t ExternalTrigConv***
- ***uint32_t ExternalTrigConvEdge***
- ***FunctionalState DMAContinuousRequests***
- ***uint32_t Overrun***
- ***uint32_t SamplingTimeCommon1***
- ***uint32_t SamplingTimeCommon2***
- ***FunctionalState OversamplingMode***
- ***ADC_OversamplingTypeDef Oversampling***
- ***uint32_t TriggerFrequencyMode***

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 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::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::ScanConvMode***
  Configure the sequencer of ADC group regular. 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.
    - 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 associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. Sequencer is automatically enabled if several channels are set (sequencer cannot be disabled, as it can be the case on other STM32 devices): If only 1 channel is set: Conversion is performed in single mode. If several channels are set: Conversions are performed in sequence mode. This parameter can be a value of ***ADC_Scan_mode***
- ***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) has been retrieved by user software, using function **HAL_ADC_GetValue()**. 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_PollForConversion()** to ensure that conversion is completed and **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion start.
- ***FunctionalState ADC_InitTypeDef::LowPowerAutoPowerOff***
  Select the auto-off mode: 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 automatic wait mode (parameter 'LowPowerAutoWait'). This parameter can be set to ENABLE or DISABLE.
- ***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 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. Note: On this STM32 series, ADC group regular number of discontinuous ranks increment is fixed to one-by-one.
- ***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***
- ***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 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 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). 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 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). 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).
- ***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 group regular.
- ***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.

7.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32c0xx_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***

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***
  Add or remove the channel from ADC regular group sequencer and specify its conversion rank. This parameter is dependent 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 (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_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.

7.1.4 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32c0xx_hal_adc.h

Data Fields

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

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 group regular. For Analog Watchdog 2 and 3: Several channels can be monitored by applying successively the AWD init structure. 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'). 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 (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: 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 final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Configures the ADC analog watchdog Low threshold value. 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: 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 final computation (after ratio and shift application): ADC data register bitfield [15:4] (12 most significant bits).

7.1.5 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32c0xx_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***

- ***uint32_t ADCGroupRegularSequencerRanks***
- ***void(* ConvCpltCallback***
- ***void(* ConvHalfCpltCallback***
- ***void(* LevelOutOfWindowCallback***
- ***void(* ErrorCallback***
- ***void(* LevelOutOfWindow2Callback***
- ***void(* LevelOutOfWindow3Callback***
- ***void(* EndOfSamplingCallback***
- ***void(* CalibrationCpltCallback***
- ***void(* ADCReadyCallback***
- ***void(* ChannelConfigReadyCallback***
- ***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
- ***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::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::ADCReadyCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Ready callback
- ***void(* __ADC_HandleTypeDef::ChannelConfigReadyCallback)(struct __ADC_HandleTypeDef *hadc)***
  ADC Channel Configuration 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

- 12-bit, 10-bit, 8-bit or 6-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 (common to group of channels)
- External trigger (timer or EXTI) with configurable polarity
- DMA request generation for transfer of conversions data of regular group.
- 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. Caution: On STM32C0, ADC clock frequency max is 35MHz (refer to device datasheet). Therefore, ADC clock source from RCC and ADC clock prescaler must be configured to remain below this maximum frequency.
   - Two clock settings are mandatory:
     - ADC clock (core clock, also possibly conversion clock).
     - ADC clock (conversions clock). Four possible clock sources: synchronous clock from APB clock (same as ADC core clock) or asynchronous clock from RCC level: SYSCLK, HSI48.
     - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory: core clock)
   - 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 __ADCx_FORCE_RESET(), __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:
     - RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI14;
     - RCC_OscInitStructure.HSI14State = RCC_HSI14_OFF; (if not used for system clock)
     - HAL_RCC_OscConfig(&RCC_OscInitStructure);
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
- 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
- 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".

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_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_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_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_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.

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.
- 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, 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.

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.

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_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, const ADC_ChannelConfTypeDef * sConfig)

Function description

Configure a channel to be assigned to ADC group regular.

Parameters

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

Return values

- **HAL:** status

Notes

- In case of usage of internal measurement channels: 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, const ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)

Function description

Configure the analog watchdog.

Parameters

- **hadc:** ADC handle
- **AnalogWDGConfig:** 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".
- On this STM32 series, analog watchdog thresholds can be modified while ADC conversion is on going. In this case, some constraints must be taken into account: the programmed threshold values are effective from the next ADC EOC (end of unitary conversion). Considering that registers write delay may happen due to bus activity, this might cause an uncertainty on the effective timing of the new programmed threshold values.

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)

Function description

Stop ADC conversion.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

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

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.

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 Mode

ADC_ANALOGWATCHDOG_NONE

ADC AWD not selected

ADC_ANALOGWATCHDOG_SINGLE_REG

ADC AWD applied to a regular group single channel

ADC_ANALOGWATCHDOG_ALL_REG

ADC AWD applied to regular group all channels

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

Notes:

- Analog watchdog 1 is available on all stm32 series Analog watchdog 2 and 3 are not available on all series ADC End of Sampling event

ADC_AWD1_EVENT

ADC Analog watchdog 1 event (main analog watchdog)

ADC_AWD2_EVENT

ADC Analog watchdog 2 event (additional analog watchdog)

ADC_AWD3_EVENT

ADC Analog watchdog 3 event (additional analog watchdog)

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_CCRDY

ADC channel configuration 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_AWD1

ADC Analog watchdog 1 flag

ADC_FLAG_AWD2

ADC Analog watchdog 2 flag

ADC_FLAG_AWD3

ADC Analog watchdog 3 flag

ADC_FLAG_EOCAL

ADC End of Calibration interrupt 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

External channel (GPIO pin) ADCx_IN0

ADC_CHANNEL_1

External channel (GPIO pin) ADCx_IN1

ADC_CHANNEL_2

External channel (GPIO pin) ADCx_IN2

ADC_CHANNEL_3

External channel (GPIO pin) ADCx_IN3

ADC_CHANNEL_4

External channel (GPIO pin) ADCx_IN4

ADC_CHANNEL_5

External channel (GPIO pin) ADCx_IN5

ADC_CHANNEL_6

External channel (GPIO pin) ADCx_IN6

ADC_CHANNEL_7

External channel (GPIO pin) ADCx_IN7

ADC_CHANNEL_8

External channel (GPIO pin) ADCx_IN8

ADC_CHANNEL_9

External channel (GPIO pin) ADCx_IN9

ADC_CHANNEL_10

External channel (GPIO pin) ADCx_IN10

ADC_CHANNEL_11

External channel (GPIO pin) ADCx_IN11

ADC_CHANNEL_12

External channel (GPIO pin) ADCx_IN12

ADC_CHANNEL_13

External channel (GPIO pin) ADCx_IN13

ADC_CHANNEL_14

External channel (GPIO pin) ADCx_IN14

ADC_CHANNEL_15

External channel (GPIO pin) ADCx_IN15

ADC_CHANNEL_16

External channel (GPIO pin) ADCx_IN16

ADC_CHANNEL_17

External channel (GPIO pin) ADCx_IN17

ADC_CHANNEL_18

External channel (GPIO pin) ADCx_IN18

ADC_CHANNEL_19

External channel (GPIO pin) ADCx_IN19

ADC_CHANNEL_20

External channel (GPIO pin) ADCx_IN20

ADC_CHANNEL_21

External channel (GPIO pin) ADCx_IN21

ADC_CHANNEL_22

External channel (GPIO pin) ADCx_IN22

ADC_CHANNEL_VREFINT

Internal channel VrefInt: Internal voltage reference.

ADC_CHANNEL_TEMPSENSOR

Internal channel Temperature sensor.

ADC_CHANNEL_VDDA

Internal channel VDDA

ADC_CHANNEL_VSSA

Internal channel VSSA

Channel - Sampling time

ADC_SAMPLETIME_1CYCLE_5

Sampling time 1.5 ADC clock cycle

ADC_SAMPLETIME_3CYCLES_5

Sampling time 3.5 ADC clock cycles

ADC_SAMPLETIME_7CYCLES_5

Sampling time 7.5 ADC clock cycles

ADC_SAMPLETIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

ADC_SAMPLETIME_19CYCLES_5

Sampling time 19.5 ADC clock cycles

ADC_SAMPLETIME_39CYCLES_5

Sampling time 39.5 ADC clock cycles

ADC_SAMPLETIME_79CYCLES_5

Sampling time 79.5 ADC clock cycles

ADC_SAMPLETIME_160CYCLES_5

Sampling time 160.5 ADC clock cycles

ADC common - Clock source

ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock from AHB clock without prescaler. This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle)

ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock from AHB clock with prescaler division by 2

ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock from AHB clock with prescaler division by 4

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)

Oversampling - Discontinuous mode

ADC_TRIGGEREDMODE_SINGLE_TRIGGER

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

ADC_TRIGGEREDMODE_MULTI_TRIGGER

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

Oversampling - Ratio

ADC_OVERSAMPLING_RATIO_2

Notes:

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

ADC_OVERSAMPLING_RATIO_4

ADC oversampling ratio 4

ADC_OVERSAMPLING_RATIO_8

ADC oversampling ratio 8

ADC_OVERSAMPLING_RATIO_16

ADC oversampling ratio 16

ADC_OVERSAMPLING_RATIO_32

ADC oversampling ratio 32

ADC_OVERSAMPLING_RATIO_64

ADC oversampling ratio 64

ADC_OVERSAMPLING_RATIO_128

ADC oversampling ratio 128

ADC_OVERSAMPLING_RATIO_256

ADC oversampling ratio 256

Oversampling - Data shift

ADC_RIGHTBITSHIFT_NONE

Notes:

- The sum of the ADC conversions data is divided by "Rightbitshift" number to result as the ADC oversampling conversion data) ADC oversampling no shift

ADC_RIGHTBITSHIFT_1

ADC oversampling right shift of 1 ranks

ADC_RIGHTBITSHIFT_2

ADC oversampling right shift of 2 ranks

ADC_RIGHTBITSHIFT_3

ADC oversampling right shift of 3 ranks

ADC_RIGHTBITSHIFT_4

ADC oversampling right shift of 4 ranks

ADC_RIGHTBITSHIFT_5

ADC oversampling right shift of 5 ranks

ADC_RIGHTBITSHIFT_6

ADC oversampling right shift of 6 ranks

ADC_RIGHTBITSHIFT_7

ADC oversampling right shift of 7 ranks

ADC_RIGHTBITSHIFT_8

ADC oversampling right shift of 8 ranks

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_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, ...)

ADC_RANK_NONE

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

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 group regular - Trigger frequency mode

ADC_TRIGGER_FREQ_HIGH

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". Trigger frequency mode set to high frequency.

ADC_TRIGGER_FREQ_LOW

Trigger frequency mode set to low frequency.

ADC instance - Resolution

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

ADC instance - Sampling time common to a group of channels

ADC_SAMPLINGTIME_COMMON_1

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

ADC_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_CCRDY ADC channel configuration 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_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)

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_CCRDY ADC channel configuration 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_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)

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_CCRDY ADC channel configuration 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_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)

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_CCRDY ADC channel configuration 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_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)

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_CCRDY ADC channel configuration 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_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)

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
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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
  - ADC_CHANNEL_1
  - ADC_CHANNEL_2
  - ADC_CHANNEL_3
  - ADC_CHANNEL_4
  - ADC_CHANNEL_5
  - ADC_CHANNEL_6
  - ADC_CHANNEL_7
  - ADC_CHANNEL_8 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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
  - ADC_CHANNEL_19
  - ADC_CHANNEL_20
  - ADC_CHANNEL_21
  - ADC_CHANNEL_22

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
  - ADC_CHANNEL_TEMPSENSOR
  - ADC_CHANNEL_VDDA
  - ADC_CHANNEL_VSSA

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_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_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:

- __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_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B
- __ADC_RESOLUTION_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_RESOLUTION_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

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:

- __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_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

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_VREFANALOG_VOLTAGE

Description:

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

Parameters:

- __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_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

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_TYP_PARAMS

Description:

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

Parameters:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On this STM32 series, 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 this STM32 series, 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_12B
  - ADC_RESOLUTION_10B
  - ADC_RESOLUTION_8B
  - ADC_RESOLUTION_6B

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. : On this STM32 series, calibtation parameter TS_CAL1 can be used to improve calculation accuracy. Refer to TEMPSENSOR_CAL1_ADDR. 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

ADC interrupts definition

ADC_IT_RDY

ADC Ready interrupt source

ADC_IT_CCRDY

ADC channel configuration 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_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 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_TRGO2

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

ADC_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).

ADC_EXTERNALTRIG_T2_TRGO

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

ADC_EXTERNALTRIG_T3_TRGO

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

ADC_EXTERNALTRIG_T15_TRGO

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

ADC_EXTERNALTRIG_EXT_IT11

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

ADC Regular Conversion Number Verification

IS_ADC_REGULAR_NB_CONV

ADC sequencer scan mode

ADC_SCAN_DISABLE

Sequencer set to fully configurable: only the rank 1 is enabled (no scan sequence on several ranks)

ADC_SCAN_ENABLE

Sequencer set to fully configurable: sequencer length and each rank affectation to a channel are configurable.

ADC_SCAN_SEQ_FIXED

Sequencer set to not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Scan direction forward: from channel 0 to channel 18

ADC_SCAN_SEQ_FIXED_BACKWARD

Sequencer set to not fully configurable: sequencer length and each rank affectation to a channel are fixed by channel HW number (channel 0 fixed on rank 0, channel 1 fixed on rank1, ...). Scan direction backward: from channel 18 to channel 0

ADC_SCAN_DIRECTION_FORWARD

ADC_SCAN_DIRECTION_BACKWARD

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 from internal process (ex : 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

Not available on this STM32 series: A conversion on 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

Not available on this STM32 series: Conversion data available on group injected

HAL_ADC_STATE_INJ_JQOVF

Not available on this STM32 series: Injected queue overflow occurrence

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog 1

HAL_ADC_STATE_AWD2

Not available on this STM32 series: Out-of-window occurrence of ADC analog watchdog 2

HAL_ADC_STATE_AWD3

Not available on this STM32 series: Out-of-window occurrence of ADC analog watchdog 3

HAL_ADC_STATE_MULTIMODE_SLAVE

Not available on this STM32 series: ADC in multimode slave state, controlled by another ADC master (when feature available)

8 HAL ADC Extension Driver

8.1 ADCEx Firmware driver API description

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

8.1.1 IO operation functions

This section provides functions allowing to:

- Perform the ADC self-calibration.
- Get calibration factors.
- Set calibration factors.

This section contains the following APIs:

- *HAL_ADCEx_Calibration_Start()*
- *HAL_ADCEx_Calibration_GetValue()*
- *HAL_ADCEx_Calibration_SetValue()*
- *HAL_ADCEx_LevelOutOfWindow2Callback()*
- *HAL_ADCEx_LevelOutOfWindow3Callback()*
- *HAL_ADCEx_EndOfSamplingCallback()*
- *HAL_ADCEx_ChannelConfigReadyCallback()*

8.1.2 Detailed description of functions

HAL_ADCEx_Calibration_Start

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef * hadc)

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

Return values

- **HAL:** status

Notes

- Calibration factor can be read after calibration, using function HAL_ADC_GetValue() (value on 7 bits: from DR[6;0]).

HAL_ADCEx_Calibration_GetValue

Function name

uint32_t HAL_ADCEx_Calibration_GetValue (const ADC_HandleTypeDef * hadc)

Function description

Get the calibration factor.

Parameters

- **hadc:** ADC handle.

Return values

- **Calibration:** value.

HAL_ADCEx_Calibration_SetValue

Function name

HAL_StatusTypeDef HAL_ADCEx_Calibration_SetValue (ADC_HandleTypeDef * hadc, uint32_t CalibrationFactor)

Function description

Set the calibration factor to overwrite automatic conversion result.

Parameters

- **hadc:** ADC handle
- **CalibrationFactor:** Calibration factor (coded on 7 bits maximum)

Return values

- **HAL:** state

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_ChannelConfigReadyCallback

Function name

void HAL_ADCEx_ChannelConfigReadyCallback (ADC_HandleTypeDef * hadc)

Function description

ADC channel configuration ready callback in non-blocking mode.

Parameters

- **hadc:** ADC handle

Return values

- **None:**

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.

8.2 ADCEx Firmware driver defines

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

8.2.1 ADCEx

ADCEx

9 HAL CORTEX Generic Driver

9.1 CORTEX Firmware driver registers structures

9.1.1 MPU_Region_InitTypeDef

MPU_Region_InitTypeDef is defined in the stm32c0xx_hal_cortex.h

Data Fields

- ***uint8_t Enable***
- ***uint8_t Number***
- ***uint32_t BaseAddress***
- ***uint8_t Size***
- ***uint8_t SubRegionDisable***
- ***uint8_t TypeExtField***
- ***uint8_t AccessPermission***
- ***uint8_t DisableExec***
- ***uint8_t IsShareable***
- ***uint8_t IsCacheable***
- ***uint8_t IsBufferable***

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 number 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.
- ***uint8_t MPU_Region_InitTypeDef::Size***
  Specifies the size of the region to protect. This parameter can be a value of ***CORTEX_MPU_Region_Size***
- ***uint8_t MPU_Region_InitTypeDef::SubRegionDisable***
  Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint8_t MPU_Region_InitTypeDef::TypeExtField***
  Specifies the TEX field level. This parameter can be a value of ***CORTEX_MPU_TEX_Levels***
- ***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***
- ***uint8_t MPU_Region_InitTypeDef::IsCacheable***
  Specifies the cacheable status of the region protected. This parameter can be a value of ***CORTEX_MPU_Access_Cacheable***
- ***uint8_t MPU_Region_InitTypeDef::IsBufferable***
  Specifies the bufferable status of the protected region. This parameter can be a value of ***CORTEX_MPU_Access_Bufferable***

9.2 CORTEX Firmware driver API description

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

9.2.1 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 M0+ exceptions are managed by CMSIS functions.

1. Enable and Configure the priority of the selected IRQ Channels. The priority can be 0..3.

Note: Lower priority values gives higher priority.

Note: Priority Order:

1. *Lowest priority.*
2. *Lowest hardware priority (IRQn position).*

2. Configure the priority of the selected IRQ Channels using HAL_NVIC_SetPriority()
3. Enable the selected IRQ Channels using HAL_NVIC_EnableIRQ()

Note: Negative value of IRQn_Type are not allowed.

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 (0x03).
  - 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 stm32c0xx_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

9.2.2 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_SetPriority()***
- ***HAL_NVIC_EnableIRQ()***
- ***HAL_NVIC_DisableIRQ()***
- ***HAL_NVIC_SystemReset()***
- ***HAL_SYSTICK_Config()***

9.2.3 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_GetPriority()***
- ***HAL_NVIC_SetPendingIRQ()***

- *HAL_NVIC_GetPendingIRQ()*
- *HAL_NVIC_ClearPendingIRQ()*
- *HAL_SYSTICK_CLKSourceConfig()*
- *HAL_SYSTICK_IRQHandler()*
- *HAL_SYSTICK_Callback()*
- *HAL_MPU_Enable()*
- *HAL_MPU_Disable()*
- *HAL_MPU_EnableRegion()*
- *HAL_MPU_DisableRegion()*
- *HAL_MPU_ConfigRegion()*

9.2.4 Detailed description of functions

HAL_NVIC_SetPriority

Function name

void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)

Function description

Sets 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 stm32c0xx.h file)
- **PreemptPriority:** The preemption priority for the IRQn channel. This parameter can be a value between 0 and 3. A lower priority value indicates a higher priority
- **SubPriority:** the subpriority level for the IRQ channel. with stm32c0xx devices, this parameter is a dummy value and it is ignored, because no subpriority supported in Cortex M0+ based products.

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 (stm32c0xxxx.h))

Return values

- **None:**

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 (stm32c0xxxx.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_GetPriority

Function name

uint32_t HAL_NVIC_GetPriority (IRQn_Type IRQn)

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 (stm32c0xxxx.h))

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 (stm32c0xxxx.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 (stm32c0xxxx.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 (stm32c0xxxx.h))

Return values

- **None:**

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_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source.
  - SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.

Return values

- **None:**

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.

Return values

- **None:**

HAL_MPU_DisableRegion

Function name

void HAL_MPU_DisableRegion (uint32_t RegionNumber)

Function description

Disable the MPU Region.

Return values

- **None:**

HAL_MPU_ConfigRegion

Function name

void HAL_MPU_ConfigRegion (const MPU_Region_InitTypeDef * MPU_Init)

Function description

Initialize and configure the Region and the memory to be protected.

Parameters

- **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- **None:**

9.3 CORTEX Firmware driver defines

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

9.3.1 CORTEX

CORTEX

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE

MPU_ACCESS_NOT_SHAREABLE

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_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB

MPU_REGION_SIZE_256KB

MPU_REGION_SIZE_512KB

MPU_REGION_SIZE_1MB

MPU_REGION_SIZE_2MB

MPU_REGION_SIZE_4MB

MPU_REGION_SIZE_8MB

MPU_REGION_SIZE_16MB

MPU_REGION_SIZE_32MB

MPU_REGION_SIZE_64MB

MPU_REGION_SIZE_128MB

MPU_REGION_SIZE_256MB

MPU_REGION_SIZE_512MB

MPU_REGION_SIZE_1GB

MPU_REGION_SIZE_2GB

MPU_REGION_SIZE_4GB

CORTEX MPU TEX Levels

MPU_TEX_LEVEL0

MPU_TEX_LEVEL1

MPU_TEX_LEVEL2

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

10 HAL CRC Generic Driver

10.1 CRC Firmware driver registers structures

10.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32c0xx_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

10.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32c0xx_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

10.2 CRC Firmware driver API description

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

10.2.1 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

10.2.2 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()***

10.2.3 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()***

10.2.4 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()***

10.2.5 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

10.3 CRC Firmware driver defines

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

10.3.1 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

11 HAL CRC Extension Driver

11.1 CRCEx Firmware driver API description

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

11.1.1 How to use this driver

- Set user-defined generating polynomial through HAL_CRCEx_Polynomial_Set()
- Configure Input or Output data inversion

11.1.2 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()***

11.1.3 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

11.2 CRCEx Firmware driver defines

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

11.2.1 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

12 HAL DMA Generic Driver

12.1 DMA Firmware driver registers structures

12.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32c0xx_hal_dma.h

Data Fields

- ***uint32_t Request***
- ***uint32_t Direction***
- ***uint32_t PeriphInc***
- ***uint32_t MemInc***
- ***uint32_t PeriphDataAlignment***
- ***uint32_t MemDataAlignment***
- ***uint32_t Mode***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t DMA_InitTypeDef::Request***
  Specifies the request selected for the specified channel. This parameter can be a value of ***DMA_request***
- ***uint32_t DMA_InitTypeDef::Direction***
  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_Data_transfer_direction***
- ***uint32_t DMA_InitTypeDef::PeriphInc***
  Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of ***DMA_Peripheral_incremented_mode***
- ***uint32_t DMA_InitTypeDef::MemInc***
  Specifies whether the memory address register should be incremented or not. This parameter can be a value of ***DMA_Memory_incremented_mode***
- ***uint32_t DMA_InitTypeDef::PeriphDataAlignment***
  Specifies the Peripheral data width. This parameter can be a value of ***DMA_Peripheral_data_size***
- ***uint32_t DMA_InitTypeDef::MemDataAlignment***
  Specifies the Memory data width. This parameter can be a value of ***DMA_Memory_data_size***
- ***uint32_t DMA_InitTypeDef::Mode***
  Specifies the operation mode of the DMAy Channelx. This parameter can be a value of ***DMA_mode***
  Note:
  – The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- ***uint32_t DMA_InitTypeDef::Priority***
  Specifies the software priority for the DMAy Channelx. This parameter can be a value of ***DMA_Priority_level***

12.1.2 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32c0xx_hal_dma.h

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_DMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***

- ***void(* XferAbortCallback***
- ***__IO uint32_t ErrorCode***
- ***uint32_t ChannelIndex***
- ***DMAMUX_Channel_TypeDef * DMAmuxChannel***
- ***DMAMUX_ChannelStatus_TypeDef * DMAmuxChannelStatus***
- ***uint32_t DMAmuxChannelStatusMask***
- ***DMAMUX_RequestGen_TypeDef * DMAmuxRequestGen***
- ***DMAMUX_RequestGenStatus_TypeDef * DMAmuxRequestGenStatus***
- ***uint32_t DMAmuxRequestGenStatusMask***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
  Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
  DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
  DMA locking object
- ***__IO HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
  DMA transfer state
- ***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
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
  DMA Error code
- ***uint32_t __DMA_HandleTypeDef::ChannelIndex***
  DMA Channel Index
- ***DMAMUX_Channel_TypeDef* __DMA_HandleTypeDef::DMAmuxChannel***
  Register base address
- ***DMAMUX_ChannelStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxChannelStatus***
  DMAMUX Channels Status Base Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxChannelStatusMask***
  DMAMUX Channel Status Mask
- ***DMAMUX_RequestGen_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGen***
  DMAMUX request generator Base Address
- ***DMAMUX_RequestGenStatus_TypeDef* __DMA_HandleTypeDef::DMAmuxRequestGenStatus***
  DMAMUX request generator Address
- ***uint32_t __DMA_HandleTypeDef::DMAmuxRequestGenStatusMask***
  DMAMUX request generator Status mask

12.2 DMA Firmware driver API description

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

12.2.1 How to use this driver

1. 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 the Reference manual for connection between peripherals and DMA requests.

2. For a given Channel, program the required configuration through the following parameters: Channel request, Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode using HAL_DMA_Init() function. Prior to HAL_DMA_Init the CLK shall be enabled for both DMA & DMAMUX thanks to: DMA1 : __HAL_RCC_DMA1_CLK_ENABLE();
3. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
4. Use HAL_DMA_Abort() function to abort the current transfer

Note: In Memory-to-Memory transfer mode, Circular mode is not allowed.

Polling mode IO operation

- Use HAL_DMA_Start() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use HAL_DMA_PollForTransfer() to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

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_Start_IT() to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use HAL_DMA_IRQHandler() called under DMA_IRQHandler() Interrupt subroutine
- At the end of data transfer HAL_DMA_IRQHandler() function is executed and user can add his own function by customization of function pointer XferCpltCallback and XferErrorCallback (i.e. a member of DMA handle structure).

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 DMA HAL driver header file for more useful macros

12.2.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Channel source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Channel priority value.

The HAL_DMA_Init() function follows the DMA configuration procedures as described in reference manual.

This section contains the following APIs:

- ***HAL_DMA_Init()***
- ***HAL_DMA_DeInit()***

12.2.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete

- Handle DMA interrupt request

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()*

12.2.4 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- *HAL_DMA_GetState()*
- *HAL_DMA_GetError()*

12.2.5 Detailed description of functions

HAL_DMA_Init

Function name

HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)

Function description

Initialize the DMA according to the specified parameters in the DMA_InitTypeDef and initialize 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 * hdma)

Function description

DeInitialize the DMA peripheral.

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 * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function description

Start the DMA Transfer.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMA_Start_IT

Function name

HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)

Function description

Start the DMA Transfer with interrupt enabled.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
- **SrcAddress:** The source memory Buffer address
- **DstAddress:** The destination memory Buffer address
- **DataLength:** The length of data to be transferred from source to destination

Return values

- **HAL:** status

HAL_DMA_Abort

Function name

HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)

Function description

Abort the DMA Transfer.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** status

HAL_DMA_Abort_IT

Function name

HAL_StatusTypeDef HAL_DMA_Abort_IT (DMA_HandleTypeDef * hdma)

Function description

Aborts the DMA Transfer in Interrupt mode.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.

Return values

- **HAL:** status

HAL_DMA_PollForTransfer

Function name

HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, HAL_DMA_LevelCompleteTypeDef CompleteLevel, uint32_t Timeout)

Function description

Polling for transfer complete.

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 * hdma)

Function description

Handle DMA interrupt request.

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 * hdma, HAL_DMA_CallbackIDTypeDef CallbackID, void(*)(DMA_HandleTypeDef *_hdma) pCallback)

Function description

Register callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifier a HAL_DMA_CallbackIDTypeDef ENUM as parameter.
- **pCallback:** pointer to private callback function which has pointer to a DMA_HandleTypeDef structure as parameter.

Return values

- **HAL:** status

HAL_DMA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_DMA_UnRegisterCallback (DMA_HandleTypeDef * hdma, HAL_DMA_CallbackIDTypeDef CallbackID)

Function description

UnRegister callbacks.

Parameters

- **hdma:** pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Stream.
- **CallbackID:** User Callback identifier a HAL_DMA_CallbackIDTypeDef ENUM as parameter.

Return values

- **HAL:** status

HAL_DMA_GetState

Function name

HAL_DMA_StateTypeDef HAL_DMA_GetState (const DMA_HandleTypeDef * hdma)

Function description

Return the DMA handle state.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

Return values

- **HAL:** state

HAL_DMA_GetError

Function name

uint32_t HAL_DMA_GetError (const DMA_HandleTypeDef * hdma)

Function description

Return the DMA 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

12.3 DMA Firmware driver defines

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

12.3.1 DMA

DMA

DMA Data 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 Error Code

HAL_DMA_ERROR_NONE

No error

HAL_DMA_ERROR_TE

Transfer error

HAL_DMA_ERROR_NO_XFER

Abort requested with no Xfer ongoing

HAL_DMA_ERROR_TIMEOUT

Timeout error

HAL_DMA_ERROR_NOT_SUPPORTED

Not supported mode

HAL_DMA_ERROR_SYNC

DMAMUX sync overrun error

HAL_DMA_ERROR_REQGEN

DMAMUX request generator overrun error

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_TC_FLAG_INDEX

Description:

- Return the current DMA Channel transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer complete flag index.

__HAL_DMA_GET_HT_FLAG_INDEX

Description:

- Return the current DMA Channel half transfer complete flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified half transfer complete flag index.

__HAL_DMA_GET_TE_FLAG_INDEX

Description:

- Return the current DMA Channel transfer error flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__HAL_DMA_GET_GI_FLAG_INDEX

Description:

- Return the current DMA Channel Global interrupt flag.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: specified transfer error flag index.

__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_TCIFx: Transfer complete flag
  - DMA_FLAG_HTIFx: Half transfer complete flag
  - DMA_FLAG_TEIFx: Transfer error flag
  - DMA_FLAG_GIFx: Global interrupt flag Where x can be 1_7 to select the DMA Channel 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_TCIFx: Transfer complete flag
  - DMA_FLAG_HTIFx: Half transfer complete flag
  - DMA_FLAG_TEIFx: Transfer error flag
  - DMA_FLAG_GIFx: Global interrupt flag Where x can be 1_7 to select the DMA Channel 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 or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

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 enabled or disabled. This parameter can be any combination of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- None

__HAL_DMA_GET_IT_SOURCE

Description:

- Check whether the specified DMA Channel interrupt is enabled or disabled.

Parameters:

- __HANDLE__: DMA handle
- __INTERRUPT__: specifies the DMA interrupt source to check. This parameter can be one of the following values:
  - DMA_IT_TC: Transfer complete interrupt mask
  - DMA_IT_HT: Half transfer complete interrupt mask
  - DMA_IT_TE: Transfer error interrupt mask

Return value:

- The: state of DMA_IT (SET or RESET).

__HAL_DMA_GET_COUNTER

Description:

- Returns the number of remaining data units in the current DMA Channel transfer.

Parameters:

- __HANDLE__: DMA handle

Return value:

- The: number of remaining data units in the current DMA Channel transfer.

DMA flag definitions

DMA_FLAG_GI1

DMA_FLAG_TC1

DMA_FLAG_HT1

DMA_FLAG_TE1

DMA_FLAG_GI2

DMA_FLAG_TC2

DMA_FLAG_HT2

DMA_FLAG_TE2

DMA_FLAG_GI3

DMA_FLAG_TC3

DMA_FLAG_HT3

DMA_FLAG_TE3

DMA_FLAG_GI4

DMA_FLAG_TC4

DMA_FLAG_HT4

DMA_FLAG_TE4

DMA_FLAG_GI5

DMA_FLAG_TC5

DMA_FLAG_HT5

DMA_FLAG_TE5

DMA_FLAG_GI6

DMA_FLAG_TC6

DMA_FLAG_HT6

DMA_FLAG_TE6

DMA_FLAG_GI7

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

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 definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE

Memory data alignment : Byte

DMA_MDATAALIGN_HALFWORD

Memory data alignment : HalfWord

DMA_MDATAALIGN_WORD

Memory data alignment : Word

DMA Memory incremented mode

DMA_MINC_ENABLE

Memory increment mode Enable

DMA_MINC_DISABLE

Memory increment mode Disable

DMA mode

DMA_NORMAL

Normal mode

DMA_CIRCULAR

Circular mode

DMA Peripheral data size

DMA_PDATAALIGN_BYTE

Peripheral data alignment : Byte

DMA_PDATAALIGN_HALFWORD

Peripheral data alignment : HalfWord

DMA_PDATAALIGN_WORD

Peripheral data alignment : Word

DMA Peripheral incremented mode

DMA_PINC_ENABLE

Peripheral increment mode Enable

DMA_PINC_DISABLE

Peripheral increment mode Disable

DMA Priority level

DMA_PRIORITY_LOW

Priority level : Low

DMA_PRIORITY_MEDIUM

Priority level : Medium

DMA_PRIORITY_HIGH

Priority level : High

DMA_PRIORITY_VERY_HIGH

Priority level : Very_High

DMA request

DMA_REQUEST_MEM2MEM

memory to memory transfer

DMA_REQUEST_GENERATOR0

DMAMUX request generator 0

DMA_REQUEST_GENERATOR1

DMAMUX request generator 1

DMA_REQUEST_GENERATOR2

DMAMUX request generator 2

DMA_REQUEST_GENERATOR3

DMAMUX request generator 3

DMA_REQUEST_ADC1

DMAMUX ADC1 request

DMA_REQUEST_I2C1_RX

DMAMUX I2C1 RX request

DMA_REQUEST_I2C1_TX

DMAMUX I2C1 TX request

DMA_REQUEST_I2C2_RX

DMAMUX I2C2 RX request

DMA_REQUEST_I2C2_TX

DMAMUX I2C2 TX request

DMA_REQUEST_SPI1_RX

DMAMUX SPI1 RX request

DMA_REQUEST_SPI1_TX

DMAMUX SPI1 TX request

DMA_REQUEST_SPI2_RX

DMAMUX SPI2 RX request

DMA_REQUEST_SPI2_TX

DMAMUX SPI2 TX request

DMA_REQUEST_TIM1_CH1

DMAMUX TIM1 CH1 request

DMA_REQUEST_TIM1_CH2

DMAMUX TIM1 CH2 request

DMA_REQUEST_TIM1_CH3

DMAMUX TIM1 CH3 request

DMA_REQUEST_TIM1_CH4

DMAMUX TIM1 CH4 request

DMA_REQUEST_TIM1_TRIG_COM

DMAMUX TIM1 TRIG COM request

DMA_REQUEST_TIM1_UP

DMAMUX TIM1 UP request

DMA_REQUEST_TIM2_CH1

DMAMUX TIM2 CH1 request

DMA_REQUEST_TIM2_CH2

DMAMUX TIM2 CH2 request

DMA_REQUEST_TIM2_CH3

DMAMUX TIM2 CH3 request

DMA_REQUEST_TIM2_CH4

DMAMUX TIM2 CH4 request

DMA_REQUEST_TIM2_TRIG

DMAMUX TIM2 TRIG request

DMA_REQUEST_TIM2_UP

DMAMUX TIM2 UP request

DMA_REQUEST_TIM3_CH1

DMAMUX TIM3 CH1 request

DMA_REQUEST_TIM3_CH2

DMAMUX TIM3 CH2 request

DMA_REQUEST_TIM3_CH3

DMAMUX TIM3 CH3 request

DMA_REQUEST_TIM3_CH4

DMAMUX TIM3 CH4 request

DMA_REQUEST_TIM3_TRIG

DMAMUX TIM3 TRIG request

DMA_REQUEST_TIM3_UP

DMAMUX TIM3 UP request

DMA_REQUEST_TIM15_CH1

DMAMUX TIM15 CH1 request

DMA_REQUEST_TIM15_CH2

DMAMUX TIM15 CH2 request

DMA_REQUEST_TIM15_TRIG_COM

DMAMUX TIM15 TRIG COM request

DMA_REQUEST_TIM15_UP

DMAMUX TIM15 UP request

DMA_REQUEST_TIM16_CH1

DMAMUX TIM16 CH1 request

DMA_REQUEST_TIM16_TRIG_COM

DMAMUX TIM16 TRIG COM request

DMA_REQUEST_TIM16_UP

DMAMUX TIM16 UP request

DMA_REQUEST_TIM17_CH1

DMAMUX TIM17 CH2 request

DMA_REQUEST_TIM17_TRIG_COM

DMAMUX TIM17 TRIG COM request

DMA_REQUEST_TIM17_UP

DMAMUX TIM17 UP request

DMA_REQUEST_USART1_RX

DMAMUX USART1 RX request

DMA_REQUEST_USART1_TX

DMAMUX USART1 TX request

DMA_REQUEST_USART2_RX

DMAMUX USART2 RX request

DMA_REQUEST_USART2_TX

DMAMUX USART2 TX request

DMA_REQUEST_USART3_RX

DMAMUX USART3 RX request

DMA_REQUEST_USART3_TX

DMAMUX USART3 TX request

DMA_REQUEST_USART4_RX

DMAMUX USART4 RX request

DMA_REQUEST_USART4_TX

DMAMUX USART4 TX request

13 HAL DMA Extension Driver

13.1 DMAEx Firmware driver registers structures

13.1.1 HAL_DMA_MuxSyncConfigTypeDef

HAL_DMA_MuxSyncConfigTypeDef is defined in the stm32c0xx_hal_dma_ex.h

Data Fields

- ***uint32_t SyncSignalID***
- ***uint32_t SyncPolarity***
- ***FunctionalState SyncEnable***
- ***FunctionalState EventEnable***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncSignalID***
  Specifies the synchronization signal gating the DMA request in periodic mode. This parameter can be a value of ***DMAEx_DMAMUX_SyncSignalID_selection***
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::SyncPolarity***
  Specifies the polarity of the signal on which the DMA request is synchronized. This parameter can be a value of ***DMAEx_DMAMUX_SyncPolarity_selection***
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::SyncEnable***
  Specifies if the synchronization shall be enabled or disabled This parameter can take the value ENABLE or DISABLE
- ***FunctionalState HAL_DMA_MuxSyncConfigTypeDef::EventEnable***
  Specifies if an event shall be generated once the RequestNumber is reached. This parameter can take the value ENABLE or DISABLE
- ***uint32_t HAL_DMA_MuxSyncConfigTypeDef::RequestNumber***
  Specifies the number of DMA request that will be authorized after a sync event This parameter must be a number between Min_Data = 1 and Max_Data = 32

13.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef

HAL_DMA_MuxRequestGeneratorConfigTypeDef is defined in the stm32c0xx_hal_dma_ex.h

Data Fields

- ***uint32_t SignalID***
- ***uint32_t Polarity***
- ***uint32_t RequestNumber***

Field Documentation

- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::SignalID***
  Specifies the ID of the signal used for DMAMUX request generator This parameter can be a value of ***DMAEx_DMAMUX_SignalGeneratorID_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::Polarity***
  Specifies the polarity of the signal on which the request is generated. This parameter can be a value of ***DMAEx_DMAMUX_RequestGeneneratorPolarity_selection***
- ***uint32_t HAL_DMA_MuxRequestGeneratorConfigTypeDef::RequestNumber***
  Specifies the number of DMA request that will be generated after a signal event This parameter must be a number between Min_Data = 1 and Max_Data = 32

13.2 DMAEx Firmware driver API description

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

13.2.1 How to use this driver

The DMA Extension HAL driver can be used as follows:

- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.
- To handle the DMAMUX Interrupts, the function HAL_DMAEx_MUX_IRQHandler should be called from the DMAMUX IRQ handler i.e DMAMUX1_OVR_IRQHandler. As only one interrupt line is available for all DMAMUX channels and request generators , HAL_DMAEx_MUX_IRQHandler should be called with, as parameter, the appropriate DMA handle as many as used DMAs in the user project (exception done if a given DMA is not using the DMAMUX SYNC block neither a request generator)

13.2.2 Extended features functions

This section provides functions allowing to:

- Configure the DMA_MUX Synchronization Block using HAL_DMAEx_ConfigMuxSync function.
- Configure the DMA_MUX Request Generator Block using HAL_DMAEx_ConfigMuxRequestGenerator function. Functions HAL_DMAEx_EnableMuxRequestGenerator and HAL_DMAEx_DisableMuxRequestGenerator can then be used to respectively enable/disable the request generator.

This section contains the following APIs:

- ***HAL_DMAEx_ConfigMuxSync()***
- ***HAL_DMAEx_ConfigMuxRequestGenerator()***
- ***HAL_DMAEx_EnableMuxRequestGenerator()***
- ***HAL_DMAEx_DisableMuxRequestGenerator()***
- ***HAL_DMAEx_MUX_IRQHandler()***

13.2.3 Detailed description of functions

HAL_DMAEx_ConfigMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigMuxRequestGenerator (DMA_HandleTypeDef * hdma, HAL_DMA_MuxRequestGeneratorConfigTypeDef * pRequestGeneratorConfig)

Function description

Configure the DMAMUX request generator block used by the given DMA channel (instance).

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.
- **pRequestGeneratorConfig:** Pointer to HAL_DMA_MuxRequestGeneratorConfigTypeDef : contains the request generator parameters.

Return values

- **HAL:** status

HAL_DMAEx_EnableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_EnableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Enable the DMAMUX request generator block used by the given DMA channel (instance).

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status

HAL_DMAEx_DisableMuxRequestGenerator

Function name

HAL_StatusTypeDef HAL_DMAEx_DisableMuxRequestGenerator (DMA_HandleTypeDef * hdma)

Function description

Disable the DMAMUX request generator block used by the given DMA channel (instance).

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **HAL:** status

HAL_DMAEx_ConfigMuxSync

Function name

HAL_StatusTypeDef HAL_DMAEx_ConfigMuxSync (DMA_HandleTypeDef * hdma, HAL_DMA_MuxSyncConfigTypeDef * pSyncConfig)

Function description

Configure the DMAMUX synchronization parameters for a given DMA channel (instance).

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.
- **pSyncConfig:** Pointer to HAL_DMA_MuxSyncConfigTypeDef : contains the DMAMUX synchronization parameters

Return values

- **HAL:** status

HAL_DMAEx_MUX_IRQHandler

Function name

void HAL_DMAEx_MUX_IRQHandler (DMA_HandleTypeDef * hdma)

Function description

Handles DMAMUX interrupt request.

Parameters

- **hdma:** Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA channel.

Return values

- **None:**

13.3 DMAEx Firmware driver defines

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

13.3.1 DMAEx

DMAEx

DMAMUX RequestGeneneratorPolarity selection

HAL_DMAMUX_REQ_GEN_NO_EVENT

block request generator events

HAL_DMAMUX_REQ_GEN_RISING

generate request on rising edge events

HAL_DMAMUX_REQ_GEN_FALLING

generate request on falling edge events

HAL_DMAMUX_REQ_GEN_RISING_FALLING

generate request on rising and falling edge events

DMAMUX SignalGeneratorID selection

HAL_DMAMUX1_REQ_GEN_EXTI0

Request generator Signal is EXTI0 IT

HAL_DMAMUX1_REQ_GEN_EXTI1

Request generator Signal is EXTI1 IT

HAL_DMAMUX1_REQ_GEN_EXTI2

Request generator Signal is EXTI2 IT

HAL_DMAMUX1_REQ_GEN_EXTI3

Request generator Signal is EXTI3 IT

HAL_DMAMUX1_REQ_GEN_EXTI4

Request generator Signal is EXTI4 IT

HAL_DMAMUX1_REQ_GEN_EXTI5

Request generator Signal is EXTI5 IT

HAL_DMAMUX1_REQ_GEN_EXTI6

Request generator Signal is EXTI6 IT

HAL_DMAMUX1_REQ_GEN_EXTI7

Request generator Signal is EXTI7 IT

HAL_DMAMUX1_REQ_GEN_EXTI8

Request generator Signal is EXTI8 IT

HAL_DMAMUX1_REQ_GEN_EXTI9

Request generator Signal is EXTI9 IT

HAL_DMAMUX1_REQ_GEN_EXTI10

Request generator Signal is EXTI10 IT

HAL_DMAMUX1_REQ_GEN_EXTI11

Request generator Signal is EXTI11 IT

HAL_DMAMUX1_REQ_GEN_EXTI12

Request generator Signal is EXTI12 IT

HAL_DMAMUX1_REQ_GEN_EXTI13

Request generator Signal is EXTI13 IT

HAL_DMAMUX1_REQ_GEN_EXTI14

Request generator Signal is EXTI14 IT

HAL_DMAMUX1_REQ_GEN_EXTI15

Request generator Signal is EXTI15 IT

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH0_EVT

Request generator Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH1_EVT

Request generator Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH2_EVT

Request generator Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_REQ_GEN_DMAMUX1_CH3_EVT

Request generator Signal is DMAMUX1 Channel3 Event

HAL_DMAMUX1_REQ_GEN_TIM14_OC

Request generator Signal is TIM14 OC

DMAMUX SyncPolarity selection

HAL_DMAMUX_SYNC_NO_EVENT

block synchronization events

HAL_DMAMUX_SYNC_RISING

synchronize with rising edge events

HAL_DMAMUX_SYNC_FALLING

synchronize with falling edge events

HAL_DMAMUX_SYNC_RISING_FALLING

synchronize with rising and falling edge events

DMAMUX SyncSignalID selection

HAL_DMAMUX1_SYNC_EXTI0

Synchronization Signal is EXTI0 IT

HAL_DMAMUX1_SYNC_EXTI1

Synchronization Signal is EXTI1 IT

HAL_DMAMUX1_SYNC_EXTI2

Synchronization Signal is EXTI2 IT

HAL_DMAMUX1_SYNC_EXTI3

Synchronization Signal is EXTI3 IT

HAL_DMAMUX1_SYNC_EXTI4

Synchronization Signal is EXTI4 IT

HAL_DMAMUX1_SYNC_EXTI5

Synchronization Signal is EXTI5 IT

HAL_DMAMUX1_SYNC_EXTI6

Synchronization Signal is EXTI6 IT

HAL_DMAMUX1_SYNC_EXTI7

Synchronization Signal is EXTI7 IT

HAL_DMAMUX1_SYNC_EXTI8

Synchronization Signal is EXTI8 IT

HAL_DMAMUX1_SYNC_EXTI9

Synchronization Signal is EXTI9 IT

HAL_DMAMUX1_SYNC_EXTI10

Synchronization Signal is EXTI10 IT

HAL_DMAMUX1_SYNC_EXTI11

Synchronization Signal is EXTI11 IT

HAL_DMAMUX1_SYNC_EXTI12

Synchronization Signal is EXTI12 IT

HAL_DMAMUX1_SYNC_EXTI13

Synchronization Signal is EXTI13 IT

HAL_DMAMUX1_SYNC_EXTI14

Synchronization Signal is EXTI14 IT

HAL_DMAMUX1_SYNC_EXTI15

Synchronization Signal is EXTI15 IT

HAL_DMAMUX1_SYNC_DMAMUX1_CH0_EVT

Synchronization Signal is DMAMUX1 Channel0 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH1_EVT

Synchronization Signal is DMAMUX1 Channel1 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH2_EVT

Synchronization Signal is DMAMUX1 Channel2 Event

HAL_DMAMUX1_SYNC_DMAMUX1_CH3_EVT

Synchronization Signal is DMAMUX1 Channel3 Event

HAL_DMAMUX1_SYNC_TIM14_OC

Synchronization Signal is TIM14 OC

14 HAL EXTI Generic Driver

14.1 EXTI Firmware driver registers structures

14.1.1 EXTI_HandleTypeDef

EXTI_HandleTypeDef is defined in the stm32c0xx_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

14.1.2 EXTI_ConfigTypeDef

EXTI_ConfigTypeDef is defined in the stm32c0xx_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***

14.2 EXTI Firmware driver API description

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

14.2.1 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.

14.2.2 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().

14.2.3 Configuration functions

This section contains the following APIs:

- ***HAL_EXTI_SetConfigLine()***
- ***HAL_EXTI_GetConfigLine()***
- ***HAL_EXTI_ClearConfigLine()***
- ***HAL_EXTI_RegisterCallback()***
- ***HAL_EXTI_GetHandle()***

14.2.4 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.:**

14.3 EXTI Firmware driver defines

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

14.3.1 EXTI

EXTI

EXTI GPIOSel

EXTI_GPIOA

EXTI_GPIOB

EXTI_GPIOC

EXTI_GPIOD

EXTI_GPIOF

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_26

EXTI_LINE_27

EXTI_LINE_28

EXTI_LINE_29

EXTI_LINE_30

EXTI_LINE_31

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

15 HAL FDCAN Generic Driver

15.1 FDCAN Firmware driver registers structures

15.1.1 FDCAN_InitTypeDef

FDCAN_InitTypeDef is defined in the stm32c0xx_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***

15.1.2 FDCAN_FilterTypeDef

FDCAN_FilterTypeDef is defined in the stm32c0xx_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

15.1.3 FDCAN_TxHeaderTypeDef

FDCAN_TxHeaderTypeDef is defined in the stm32c0xx_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 15.3.1: 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

15.1.4 FDCAN_RxHeaderTypeDef

FDCAN_RxHeaderTypeDef is defined in the stm32c0xx_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

15.1.5 FDCAN_TxEventFifoTypeDef

FDCAN_TxEventFifoTypeDef is defined in the stm32c0xx_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***

15.1.6 FDCAN_HpMsgStatusTypeDef

FDCAN_HpMsgStatusTypeDef is defined in the stm32c0xx_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

15.1.7 FDCAN_ProtocolStatusTypeDef

FDCAN_ProtocolStatusTypeDef is defined in the stm32c0xx_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

15.1.8 FDCAN_ErrorCountersTypeDef

FDCAN_ErrorCountersTypeDef is defined in the stm32c0xx_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

15.1.9 FDCAN_MsgRamAddressTypeDef

FDCAN_MsgRamAddressTypeDef is defined in the stm32c0xx_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

15.1.10 __FDCAN_HandleTypeDef

__FDCAN_HandleTypeDef is defined in the stm32c0xx_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

15.2 FDCAN Firmware driver API description

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

15.2.1 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

15.2.2 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()***

15.2.3 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()***

15.2.4 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()***

15.2.5 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()***

15.2.6 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()*

15.2.7 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()*

15.2.8 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

15.3 FDCAN Firmware driver defines

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

15.3.1 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

16 HAL FLASH Generic Driver

16.1 FLASH Firmware driver registers structures

16.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32c0xx_hal_flash.h

Data Fields

- ***uint32_t TypeErase***
- ***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::Page***
  Initial Flash page to erase when page erase is enabled This parameter must be a value between 0 and (FLASH_PAGE_NB - 1)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  Number of pages to be erased. This parameter must be a value between 1 and (FLASH_PAGE_NB - value of initial page)

16.1.2 FLASH_OBProgramInitTypeDef

FLASH_OBProgramInitTypeDef is defined in the stm32c0xx_hal_flash.h

Data Fields

- ***uint32_t OptionType***
- ***uint32_t WRPArea***
- ***uint32_t WRPStartOffset***
- ***uint32_t WRPEndOffset***
- ***uint32_t RDPLevel***
- ***uint32_t USERType***
- ***uint32_t USERConfig***
- ***uint32_t PCROPConfig***
- ***uint32_t PCROP1AStartAddr***
- ***uint32_t PCROP1AEndAddr***
- ***uint32_t PCROP1BStartAddr***
- ***uint32_t PCROP1BEndAddr***
- ***uint32_t BootEntryPoint***
- ***uint32_t SecSize***

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 [FLASH_PAGE_NB - 1]
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPEndOffset***
  Write protection end offset (used for OPTIONBYTE_WRP). This parameter must be a value between WRPStartOffset and [FLASH_PAGE_NB - 1]

- ***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_ENABLE***, ***FLASH_OB_USER_BOR_LEVEL***, ***FLASH_OB_USER_nRST_STOP***, ***FLASH_OB_USER_nRST_STANDBY***, ***FLASH_OB_USER_nRST_SHUTDOWN***, ***FLASH_OB_USER_IWDG_SW***, ***FLASH_OB_USER_IWDG_STOP***, ***FLASH_OB_USER_IWDG_STANDBY***, ***FLASH_OB_USER_WWDG_SW***, ***FLASH_OB_USER_SRAM_PARITY***, ***FLASH_OB_USER_nBOOT_SEL***, ***FLASH_OB_USER_nBOOT1***, ***FLASH_OB_USER_nBOOT0***, ***FLASH_OB_USER_INPUT_RESET_HOLDER***
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROPConfig***
  Configuration of the PCROP (used for OPTIONBYTE_PCROP). This parameter must be a combination of ***FLASH_OB_PCROP_ZONE*** and ***FLASH_OB_PCROP_RDP***. Note that once set, Pcrop erase on RDP level 1 regression (PCROP_RDP bit) can not be reset. It will be reset by mass erase
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROP1AStartAddr***
  PCROP Start address (used for OPTIONBYTE_PCROP). It represents first address of start block to protect. Make sure this parameter is multiple of PCROP granularity: 512 Bytes.
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROP1AEndAddr***
  PCROP End address (used for OPTIONBYTE_PCROP). It represents first address of end block to protect. Make sure this parameter is multiple of PCROP granularity: 512 Bytes.
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROP1BStartAddr***
  PCROP Start address (used for OPTIONBYTE_PCROP). It represents first address of start block to protect. Make sure this parameter is multiple of PCROP granularity: 512 Bytes.
- ***uint32_t FLASH_OBProgramInitTypeDef::PCROP1BEndAddr***
  PCROP End address (used for OPTIONBYTE_PCROP). It represents first address of end block to protect. Make sure this parameter is multiple of PCROP granularity: 512 Bytes.
- ***uint32_t FLASH_OBProgramInitTypeDef::BootEntryPoint***
  Allow to force a unique boot entry point to Flash or system Flash
- ***uint32_t FLASH_OBProgramInitTypeDef::SecSize***
  This parameter defines securable memory area width in number of pages starting from Flash base address. This parameter must be a value between [0] and [FLASH_PAGE_NB], [0] meaning no secure area defined, [1] meaning first page only protected, etc...

16.1.3 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32c0xx_hal_flash.h

Data Fields

- ***HAL_LockTypeDef Lock***
- ***uint32_t ErrorCode***
- ***uint32_t ProcedureOnGoing***
- ***uint32_t Address***
- ***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::Page***
- ***uint32_t FLASH_ProcessTypeDef::NbPagesToErase***

16.2 FLASH Firmware driver API description

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

16.2.1 FLASH peripheral features

The Flash memory interface manages CPU AHB I-Code and D-Code 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 accelerates code execution with a system of instruction prefetch and cache lines.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Option bytes programming
- Prefetch on I-Code
- 32 cache lines of 4*64 bits on I-Code

16.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory of all STM32C0xx devices.

1. Flash Memory IO Programming functions:
   - Lock and Unlock the FLASH interface using HAL_FLASH_Unlock() and HAL_FLASH_Lock() functions
   - Program functions: double word and fast program (full row programming)
   - 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_OB_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 prefetch buffer
- Enable/Disable the Instruction cache
- Reset the Instruction cache
- Enable/Disable the Flash power-down during low-power modes
- Enable/Disable the Flash interrupts
- Monitor the Flash flags status

16.2.3 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()*

16.2.4 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()*

16.2.5 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()*

16.2.6 Detailed description of functions

HAL_FLASH_Program

Function name

HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)

Function description

Program double word or fast program of a row 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.
- **Data:** Specifies the data to be programmed This parameter is the data for the double word program and the address where are stored the data for the row fast program.

Return values

- **HAL_StatusTypeDef:** HAL Status

HAL_FLASH_Program_IT

Function name

HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)

Function description

Program double word or fast program of a row 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.
- **Data:** Specifies the data to be programmed This parameter is the data for the double word program and the address where are stored the data for the row fast program.

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: 0 Page Erase: Page which has 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: 0 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_MIS FLASH Fast programming data miss error
  - HAL_FLASH_ERROR_FAST FLASH Fast programming error
  - HAL_FLASH_ERROR_RD FLASH Read Protection error (PCROP)
  - HAL_FLASH_ERROR_OPTV FLASH Option validity 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_StatusTypeDef:** HAL Status

16.3 FLASH Firmware driver defines

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

16.3.1 FLASH

FLASH

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_MIS

HAL_FLASH_ERROR_FAST

HAL_FLASH_ERROR_RD

HAL_FLASH_ERROR_OPTV

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

Return value:

- None

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- FLASH: Latency Returned value can be one of the following values :
  - FLASH_LATENCY_0 FLASH Zero wait state
  - FLASH_LATENCY_1 FLASH One wait state

__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_INSTRUCTION_CACHE_ENABLE

Description:

- Enable the FLASH instruction cache.

Return value:

- none

__HAL_FLASH_INSTRUCTION_CACHE_DISABLE

Description:

- Disable the FLASH instruction cache.

Return value:

- none

__HAL_FLASH_INSTRUCTION_CACHE_RESET

Description:

- Reset the FLASH instruction Cache.

Return value:

- None

Notes:

- This function must be used only when the Instruction Cache is disabled.

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_MISERR

FLASH Fast programming data miss error flag

FLASH_FLAG_FASTERR

FLASH Fast programming error flag

FLASH_FLAG_RDERR

FLASH PCROP read error flag

FLASH_FLAG_OPTVERR

FLASH Option validity error flag

FLASH_FLAG_BSY

FLASH Operation Busy flag

FLASH_FLAG_CFGBSY

FLASH Configuration Busy flag

FLASH_FLAG_PESD

FLASH Programming/erase operation suspended

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_RDERR PCROP Read Error Interrupt

Return value:

- none

__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_RDERR PCROP Read Error Interrupt

Return value:

- none

__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_MISERR FLASH Fast programming data miss error flag
  - FLASH_FLAG_FASTERR FLASH Fast programming error flag
  - FLASH_FLAG_RDERR FLASH PCROP read error flag
  - FLASH_FLAG_OPTVERR FLASH Option validity error flag
  - FLASH_FLAG_BSY FLASH write/erase operations in progress flag
  - FLASH_FLAG_CFGBSY FLASH configuration is busy : program or erase setting are used.
  - FLASH_FLAG_PESD FLASH Programming/erase operation suspended

Return value:

- The: new state of FLASH_FLAG (SET or RESET).

__HAL_FLASH_CLEAR_FLAG

Description:

- Clear the FLASHs 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_MISERR FLASH Fast programming data miss error flag
  - FLASH_FLAG_FASTERR FLASH Fast programming error flag
  - FLASH_FLAG_RDERR FLASH PCROP read error flag
  - FLASH_FLAG_OPTVERR FLASH Option validity error flag

Return value:

- None

FLASH Interrupts Definition

FLASH_IT_EOP

End of FLASH Operation Interrupt source

FLASH_IT_OPERR

Error Interrupt source

FLASH_IT_RDERR

PCROP Read Error Interrupt source

FLASH Keys

FLASH_KEY1

Flash key1

FLASH_KEY2

Flash key2: used with FLASH_KEY1 to unlock the FLASH registers access

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 Option Bytes PCROP On RDP Level Type

OB_PCROP_RDP_NOT_ERASE

PCROP area is not erased when the RDP level is decreased from Level 1 to Level 0

OB_PCROP_RDP_ERASE

PCROP area is erased when the RDP level is decreased from Level 1 to Level 0 (full mass erase).Once this bit is set only, it will be reset by mass erase

FLASH Option Bytes PCROP ZONE

OB_PCROP_ZONE_A

PCROP Zone A

OB_PCROP_ZONE_B

PCROP Zone B

FLASH Option Bytes Read Protection

OB_RDP_LEVEL_0

OB_RDP_LEVEL_1

OB_RDP_LEVEL_2

Warning: When enabling read protection level 2 it is no more possible to go back to level 1 or 0

FLASH Option Bytes Secure boot lock

OB_BOOT_ENTRY_FORCED_NONE

Boot entry is free

OB_BOOT_ENTRY_FORCED_FLASH

Boot entry is forced to Flash or System Flash

FLASH Option Bytes Type

OPTIONBYTE_WRP

WRP option byte configuration

OPTIONBYTE_RDP

RDP option byte configuration

OPTIONBYTE_USER

USER option byte configuration

OPTIONBYTE_PCROP

PCROP option byte configuration

OPTIONBYTE_SEC

SEC option byte configuration

OPTIONBYTE_ALL

All option byte configuration

FLASH Option Bytes User BOR enable

OB_BOR_DISABLE

BOR Reset set to default

OB_BOR_ENABLE

Use option byte to define BOR thresholds

FLASH Option Bytes User BOR Level

OB_BOR_LEVEL_FALLING_0

BOR falling level 1 with threshold around 2.0V

OB_BOR_LEVEL_FALLING_1

BOR falling level 2 with threshold around 2.2V

OB_BOR_LEVEL_FALLING_2

BOR falling level 3 with threshold around 2.5V

OB_BOR_LEVEL_FALLING_3

BOR falling level 4 with threshold around 2.8V

OB_BOR_LEVEL_RISING_0

BOR rising level 1 with threshold around 2.1V

OB_BOR_LEVEL_RISING_1

BOR rising level 2 with threshold around 2.3V

OB_BOR_LEVEL_RISING_2

BOR rising level 3 with threshold around 2.6V

OB_BOR_LEVEL_RISING_3

BOR rising level 4 with threshold around 2.9V

FLASH Option Bytes User HSE REMAP

OB_HSE_NOT_REMAPPED_ENABLE

HSE Remap enable

OB_HSE_NOT_REMAPPED_DISABLE

HSE Remap disable

FLASH Option Bytes User input reset holder bit

OB_IRH_ENABLE

Internal Reset handler enable

OB_IRH_DISABLE

Internal Reset handler disable

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 BOOT1 Type

OB_BOOT1_SRAM

Embedded SRAM is selected as boot space (if nBOOT0=0 or BOOT0_pin=1)

OB_BOOT1_SYSTEM

System memory is selected as boot space (if nBOOT0=0 or BOOT0_pin=1)

FLASH Option Bytes User Boot0 Selection

OB_BOOT0_FROM_PIN

BOOT0 signal is defined by PA14/BOOT0 pin value

OB_BOOT0_FROM_OB

BOOT0 signal is defined by nBOOT0 option bit

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 reset config bit

OB_RESET_MODE_INPUT_ONLY

Reset pin is in Reset input mode only

OB_RESET_MODE_GPIO

Reset pin is in GPIO mode mode only

OB_RESET_MODE_INPUT_OUTPUT

Reset pin is in reset input and output mode

FLASH Option Bytes User Multiple-bonding security

OB_SECURE_MUXING_ENABLE

Multiple-bonding security enable

OB_SECURE_MUXING_DISABLE

Multiple-bonding security disable

FLASH Option Bytes User SRAM parity

OB_SRAM_PARITY_ENABLE

Sram parity enable

OB_SRAM_PARITY_DISABLE

Sram parity disable

FLASH Option Bytes User Type

OB_USER_BOR_EN

BOR reset enable

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_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_HSE_NOT_REMAPPED

Remap HSE source from PF to PC

OB_USER_RAM_PARITY_CHECK

Sram parity check control

OB_USER_SECURE_MUXING_EN

Multiple-bonding security enable

OB_USER_NBOOT_SEL

Boot Selection

OB_USER_NBOOT1

nBoot1 configuration

OB_USER_NBOOT0

nBoot0 configuration

OB_USER_NRST_MODE

Reset pin configuration

OB_USER_INPUT_RESET_HOLDER

Internal reset holder enable

OB_USER_ALL

all option bits

FLASH Option Bytes User WWDG Type

OB_WWDG_HW

Hardware window watchdog

OB_WWDG_SW

Software window watchdog

FLASH WRP Area

OB_WRPAREA_ZONE_A

Flash Zone A

OB_WRPAREA_ZONE_B

Flash Zone B

FLASH Erase Type

FLASH_TYPEERASE_PAGES

Pages erase only

FLASH_TYPEERASE_MASS

Flash mass erase activation

FLASH Program Type

FLASH_TYPEPROGRAM_DOUBLEWORD

Program a double-word (64-bit) at a specified address

FLASH_TYPEPROGRAM_FAST

Fast program a 32 row double-word (64-bit) at a specified address

17 HAL FLASH Extension Driver

17.1 FLASHEx Firmware driver API description

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

17.1.1 Flash Extended features

Comparing to other previous devices, the FLASH interface for STM32C0xx devices contains the following additional features

- Capacity up to 128 Kbytes with single bank architecture supporting read-while-write capability (RWW)
- Single bank memory organization
- PCROP protection

17.1.2 How to use this driver

This driver provides functions to configure and program the FLASH memory of all STM32C0xx 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: Erase page, erase all sectors
   - 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 :
   - Set/Reset the write protection
   - Set the Read protection Level
   - Program the user Option Bytes
   - Configure the PCROP protection
   - Set Securable memory area and boot entry point
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 Securable memory area and boot entry point information
4. Enable or disable debugger usage using HAL_FLASHEx_EnableDebugger and HAL_FLASHEx_DisableDebugger.
5. Check is flash content is empty or not using HAL_FLASHEx_FlashEmptyCheck. and modify this setting (for flash loader purpose e.g.) using HAL_FLASHEx_ForceFlashEmpty.
6. Enable securable memory area protectionusing HAL_FLASHEx_EnableSecMemProtection

17.1.3 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_EnableDebugger()***
- ***HAL_FLASHEx_DisableDebugger()***
- ***HAL_FLASHEx_FlashEmptyCheck()***

- *HAL_FLASHEx_ForceFlashEmpty()*
- *HAL_FLASHEx_EnableSecMemProtection()*

17.1.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase (const 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_EnableDebugger

Function name

void HAL_FLASHEx_EnableDebugger (void)

Function description

Enable Debugger.

Return values

- **None:**

Notes

- After calling this API, flash interface allow debugger intrusion.

HAL_FLASHEx_DisableDebugger

Function name

void HAL_FLASHEx_DisableDebugger (void)

Function description

Disable Debugger.

Return values

- **None:**

Notes

- After calling this API, Debugger is disabled: it is no more possible to break, see CPU register, etc...

HAL_FLASHEx_FlashEmptyCheck

Function name

uint32_t HAL_FLASHEx_FlashEmptyCheck (void)

Function description

Flash Empty check.

Return values

- **0:** if 1st location is not programmed else

Notes

- This API checks if first location in Flash is programmed or not. This check is done once by Option Byte Loader.

HAL_FLASHEx_ForceFlashEmpty

Function name

void HAL_FLASHEx_ForceFlashEmpty (uint32_t FlashEmpty)

Function description

Force Empty check value.

Parameters

- **FlashEmpty:** this parameter can be a value of FLASHEx Empty Check

Return values

- **None:**

Notes

- Allows to modify program empty check value in order to force this infrmation in Flash Interface, for all next reset that do not launch Option Byte Loader.

HAL_FLASHEx_EnableSecMemProtection

Function name

void HAL_FLASHEx_EnableSecMemProtection (uint32_t Bank)

Function description

Securable memory area protection enable.

Parameters

- **Bank:** Select Bank to be secured. On C0, there is only 1 bank so parameter has to be set to 0.

Return values

- **None:**

Notes

- This API locks Securable memory area which is defined in SEC_SIZE option byte (that can be retrieved calling HAL_FLASHEx_OBGetConfig API and checking Secsize).
- SEC_PROT bit can only be set, it will be reset by system reset.

HAL_FLASHEx_OBProgram

Function name

HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)

Function description

Program Option bytes.

Parameters

- **pOBInit:** Pointer to an FLASH_OBProgramInitTypeDef 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 only after an option bytes launch through the call of HAL_FLASH_OB_Launch()a Power On Resetan exit from Standby or Shutdown mode.

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_OBProgramInitTypeDef structure that contains the configuration information. The fields pOBInit->WRPArea and pOBInit->PCROPConfig should indicate which area is requested for the WRP and PCROP.

Return values

- **None:**

Notes

- warning: this API only read flash register, it does not reflect any change that would have been programmed between previous Option byte loading and current call.

FLASH_PageErase

Function name

void FLASH_PageErase (uint32_t Page)

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 Flash - 1)

Return values

- **None:**

17.2 FLASHEx Firmware driver defines

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

17.2.1 FLASHEx

FLASHEx

FLASHEx Empty Check

FLASH_PROG_NOT_EMPTY

1st location in Flash is programmed

FLASH_PROG_EMPTY

1st location in Flash is empty

18 HAL GPIO Generic Driver

18.1 GPIO Firmware driver registers structures

18.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32c0xx_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 any 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***

18.2 GPIO Firmware driver API description

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

18.2.1 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 28 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.

18.2.2 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 PF0 and PF1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

18.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- ***HAL_GPIO_Init()***
- ***HAL_GPIO_DeInit()***

18.2.4 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 peripheral according to the specified parameters in the GPIO_Init.

Parameters

- **GPIOx:** where x can be (A..F) to select the GPIO peripheral for STM32C0xx family
- **pGPIO_Init:** pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.

Return values

- **None:**

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:** where x can be (A..F) to select the GPIO peripheral for STM32C0xx 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:**

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:** where x can be (A..F) to select the GPIO peripheral for STM32C0xx family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be any combination of 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:** where x can be (A..F) to select the GPIO peripheral for STM32C0xx 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 and GPIOx_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:** where x can be (A..F) to select the GPIO peripheral for STM32C0xx 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:** where x can be (A..I) to select the GPIO peripheral for STM32C0 family
- **GPIO_Pin:** specifies the pin to be toggled.

Return values

- **None:**

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:** where x can be (A..F) to select the GPIO peripheral for STM32C0xx 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 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 detection callback.

Parameters

- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

18.3 GPIO Firmware driver defines

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

18.3.1 GPIO

GPIO

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 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 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.

Parameters:

- __EXTI_LINE__: specifies the EXTI line to check. This parameter can be 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

19 HAL GPIO Extension Driver

19.1 GPIOEx Firmware driver defines

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

19.1.1 GPIOEx

GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_MCO

MCO (MCO1 and MCO2) Alternate Function mapping

GPIO_AF0_OSC32

OSC32 (By pass and Enable) Alternate Function mapping

GPIO_AF0_SWJ

SWJ (SWD) Alternate Function mapping

GPIO_AF0_SPI1

SPI1 Alternate Function mapping

GPIO_AF0_I2S1

I2S1 Alternate Function mapping

GPIO_AF0_USART1

USART1 Alternate Function mapping

GPIO_AF0_EVENTOUT

EVENTOUT Alternate Function mapping

GPIO_AF0_IR

IR Alternate Function mapping

GPIO_AF0_OSC

OSC (By pass and Enable) Alternate Function mapping

GPIO_AF0_TIM14

TIM14 Alternate Function mapping

GPIO_AF0_USART2

USART2 Alternate Function mapping

GPIO_AF0_SPI2

SPI2 Alternate Function mapping

GPIO_AF0_USART3

USART3 Alternate Function mapping

GPIO_AF1_IR

IR Alternate Function mapping

GPIO_AF1_OSC

OSC (By pass and Enable) Alternate Function mapping

GPIO_AF1_TIM1

TIM1 Alternate Function mapping

GPIO_AF1_TIM3

TIM3 Alternate Function mapping

GPIO_AF1_USART1

USART1 Alternate Function mapping

GPIO_AF1_USART2

USART2 Alternate Function mapping

GPIO_AF1_USART4

USART4 Alternate Function mapping

GPIO_AF2_TIM1

TIM1 Alternate Function mapping

GPIO_AF2_TIM14

TIM14 Alternate Function mapping

GPIO_AF2_TIM15

TIM15 Alternate Function mapping

GPIO_AF2_TIM16

TIM16 Alternate Function mapping

GPIO_AF2_TIM17

TIM17 Alternate Function mapping

GPIO_AF3_MCO2

MCO2 Alternate Function mapping

GPIO_AF3_TIM3

TIM3 Alternate Function mapping

GPIO_AF3_SPI2

SPI2 Alternate Function mapping

GPIO_AF3_TIM2

TIM2 Alternate Function mapping

GPIO_AF3_FDCAN1

FDCAN1 Alternate Function mapping

GPIO_AF4_TIM14

TIM14 Alternate Function mapping

GPIO_AF4_TIM15

TIM15 Alternate Function mapping

GPIO_AF4_USART3

USART3 Alternate Function mapping

GPIO_AF4_USART2

USART2 Alternate Function mapping

GPIO_AF4_USART1

USART1 Alternate Function mapping

GPIO_AF4_SPI2

SPI2 Alternate Function mapping

GPIO_AF4_FDCAN1

FDCAN1 Alternate Function mapping

GPIO_AF5_I2S

I2S Alternate Function mapping

GPIO_AF5_USART1

USART1 Alternate Function mapping

GPIO_AF5_TIM1

TIM1 Alternate Function mapping

GPIO_AF5_TIM16

TIM16 Alternate Function mapping

GPIO_AF5_TIM17

TIM17 Alternate Function mapping

GPIO_AF5_TIM2

TIM2 Alternate Function mapping

GPIO_AF5_TIM15

TIM15 Alternate Function mapping

GPIO_AF5_SPI2

SPI2 Alternate Function mapping

GPIO_AF5_I2C2

I2C2 Alternate Function mapping

GPIO_AF5_USART3

USART3 Alternate Function mapping

GPIO_AF5_USART4

USART4 Alternate Function mapping

GPIO_AF6_I2C1

I2C1 Alternate Function mapping

GPIO_AF6_TIM2

TIM2 Alternate Function mapping

GPIO_AF6_SPI2

SPI2 Alternate Function mapping

GPIO_AF6_I2C2

I2C2 Alternate Function mapping

GPIO_AF6_USART4

USART4 Alternate Function mapping

GPIO_AF7_EVENTOUT

EVENTOUT Alternate Function mapping

GPIO_AF7_I2C1

I2C1 Alternate Function mapping

GPIO_AF8_I2S1

I2S1 Alternate Function mapping

GPIO_AF8_SPI1

SPI1 Alternate Function mapping

GPIO_AF8_IR

IR Alternate Function mapping

GPIO_AF8_I2C2

I2C2 Alternate Function mapping

GPIO_AF8_TIM15

TIM15 Alternate Function mapping

GPIO_AF8_USART3

USART3 Alternate Function mapping

GPIO_AF8_FDCAN1

FDCAN1 Alternate Function mapping

GPIO_AF9_TIM1

TIM1 Alternate Function mapping

GPIO_AF9_USART2

USART2 Alternate Function mapping

GPIO_AF9_USART4

USART4 Alternate Function mapping

GPIO_AF9_SPI1

SPI1 Alternate Function mapping

GPIO_AF9_I2S1

I2S1 Alternate Function mapping

GPIO_AF10_TIM1

TIM1 Alternate Function mapping

GPIO_AF10_I2C1

I2C1 Alternate Function mapping

GPIO_AF10_SPI1

SPI1 Alternate Function mapping

GPIO_AF10_TIM16

TIM16 Alternate Function mapping

GPIO_AF10_TIM17

TIM17 Alternate Function mapping

GPIO_AF11_TIM1

TIM1 Alternate Function mapping

GPIO_AF11_TIM3

TIM3 Alternate Function mapping

GPIO_AF11_MCO2

MCO2 Alternate Function mapping

GPIO_AF12_TIM3

TIM3 Alternate Function mapping

GPIO_AF12_USART1

USART1 Alternate Function mapping

GPIO_AF12_SPI2

SPI2 Alternate Function mapping

GPIO_AF13_TIM14

TIM14 Alternate Function mapping

GPIO_AF13_TIM3

TIM3 Alternate Function mapping

GPIO_AF14_USART1

USART1 Alternate Function mapping

GPIO_AF14_I2C1

I2C1 Alternate Function mapping

GPIO_AF14_TIM16

TIM16 Alternate Function mapping

GPIO_AF15_EVENTOUT

EVENTOUT Alternate Function mapping

GPIO_AF15_MCO2

MCO2 Alternate Function mapping

GPIO_AF15_FDCAN1

FDCAN1 Alternate Function mapping

IS_GPIO_AF

GPIOEx Get Port Index

GPIO_GET_INDEX

20 HAL I2C Generic Driver

20.1 I2C Firmware driver registers structures

20.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32c0xx_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***

20.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32c0xx_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

20.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

20.2.1 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

20.2.2 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()*

20.2.3 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()***

20.2.4 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()***

20.2.5 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

20.3 I2C Firmware driver defines

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

20.3.1 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_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

21 HAL I2C Extension Driver

21.1 I2CEx Firmware driver API description

The following section lists the various functions of the I2CEx library.

21.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32C0xx 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

21.1.2 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_EnableFastModePlus()
   - HAL_I2CEx_DisableFastModePlus()

21.1.3 Filter Mode Functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

- ***HAL_I2CEx_ConfigAnalogFilter()***
- ***HAL_I2CEx_ConfigDigitalFilter()***

21.1.4 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()***

21.1.5 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_I2CEx_EnableFastModePlus()***
- ***HAL_I2CEx_DisableFastModePlus()***

21.1.6 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_EnableFastModePlus

Function name

void HAL_I2CEx_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be enabled only by using I2C_FASTMODEPLUS_I2C2 parameter.

HAL_I2CEx_DisableFastModePlus

Function name

void HAL_I2CEx_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the I2C Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using I2C_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be disabled only by using I2C_FASTMODEPLUS_I2C2 parameter.

21.2 I2CEx Firmware driver defines

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

21.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FMP_NOT_SUPPORTED

Fast Mode Plus not supported

I2C_FASTMODEPLUS_PA9

Enable Fast Mode Plus on PA9

I2C_FASTMODEPLUS_PA10

Enable Fast Mode Plus on PA10

I2C_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

I2C_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

I2C_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

I2C_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

I2C_FASTMODEPLUS_PC14

Enable Fast Mode Plus on PC14

I2C_FASTMODEPLUS_I2C1

Enable Fast Mode Plus on I2C1 pins

I2C_FASTMODEPLUS_I2C2

Enable Fast Mode Plus on I2C2 pins

22 HAL I2S Generic Driver

22.1 I2S Firmware driver registers structures

22.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32c0xx_hal_i2s.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t CPOL***

Field Documentation

- ***uint32_t I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_Mode***
- ***uint32_t I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_Standard***
- ***uint32_t I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_Data_Format***
- ***uint32_t I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_MCLK_Output***
- ***uint32_t I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_Audio_Frequency***
- ***uint32_t I2S_InitTypeDef::CPOL***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_Clock_Polarity***

22.1.2 __I2S_HandleTypeDef

__I2S_HandleTypeDef is defined in the stm32c0xx_hal_i2s.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***I2S_InitTypeDef Init***
- ***uint16_t * pTxBuffPtr***
- ***__IO uint16_t TxXferSize***
- ***__IO uint16_t TxXferCount***
- ***uint16_t * pRxBuffPtr***
- ***__IO uint16_t RxXferSize***
- ***__IO uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***__IO HAL_LockTypeDef Lock***
- ***__IO HAL_I2S_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* TxCpltCallback***
- ***void(* RxCpltCallback***
- ***void(* TxHalfCpltCallback***
- ***void(* RxHalfCpltCallback***
- ***void(* ErrorCallback***

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

Field Documentation

- ***SPI_TypeDef* __I2S_HandleTypeDef::Instance***
  I2S registers base address
- ***I2S_InitTypeDef __I2S_HandleTypeDef::Init***
  I2S communication parameters
- ***uint16_t* __I2S_HandleTypeDef::pTxBuffPtr***
  Pointer to I2S Tx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferSize***
  I2S Tx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::TxXferCount***
  I2S Tx transfer Counter
- ***uint16_t* __I2S_HandleTypeDef::pRxBuffPtr***
  Pointer to I2S Rx transfer buffer
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferSize***
  I2S Rx transfer size
- ***__IO uint16_t __I2S_HandleTypeDef::RxXferCount***
  I2S Rx transfer counter (This field is initialized at the same value as transfer size at the beginning of the transfer and decremented when a sample is received NbSamplesReceived = RxBufferSize-RxBufferCount)
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmatx***
  I2S Tx DMA handle parameters
- ***DMA_HandleTypeDef* __I2S_HandleTypeDef::hdmarx***
  I2S Rx DMA handle parameters
- ***__IO HAL_LockTypeDef __I2S_HandleTypeDef::Lock***
  I2S locking object
- ***__IO HAL_I2S_StateTypeDef __I2S_HandleTypeDef::State***
  I2S communication state
- ***__IO uint32_t __I2S_HandleTypeDef::ErrorCode***
  I2S Error code This parameter can be a value of ***I2S_Error***
- ***void(* __I2S_HandleTypeDef::TxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Completed callback
- ***void(* __I2S_HandleTypeDef::RxCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Completed callback
- ***void(* __I2S_HandleTypeDef::TxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Tx Half Completed callback
- ***void(* __I2S_HandleTypeDef::RxHalfCpltCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Rx Half Completed callback
- ***void(* __I2S_HandleTypeDef::ErrorCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Error callback
- ***void(* __I2S_HandleTypeDef::MspInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp Init callback
- ***void(* __I2S_HandleTypeDef::MspDeInitCallback)(struct __I2S_HandleTypeDef *hi2s)***
  I2S Msp DeInit callback

22.2 I2S Firmware driver API description

The following section lists the various functions of the I2S library.

22.2.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a I2S_HandleTypeDef handle structure.

2. Initialize the I2S low level resources by implement the HAL_I2S_MspInit() API:
   a. Enable the SPIx interface clock.
   b. I2S pins configuration:
      - Enable the clock for the I2S GPIOs.
      - Configure these I2S pins as alternate function pull-up.
   c. NVIC configuration if you need to use interrupt process (HAL_I2S_Transmit_IT() and HAL_I2S_Receive_IT() APIs).
      - Configure the I2Sx interrupt priority.
      - Enable the NVIC I2S IRQ handle.
   d. DMA Configuration if you need to use DMA process (HAL_I2S_Transmit_DMA() and HAL_I2S_Receive_DMA() APIs:
      - Declare a DMA handle structure for the Tx/Rx Stream/Channel.
      - Enable the DMAx interface clock.
      - Configure the declared DMA handle structure with the required Tx/Rx parameters.
      - Configure the DMA Tx/Rx Stream/Channel.
      - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
      - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Stream/Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using HAL_I2S_Init() function.

Note: The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_I2S_ENABLE_IT() and __HAL_I2S_DISABLE_IT() inside the transmit and receive process.

Note: Make sure that either:

- *SYSCLK is selected as I2S1 clock or*
- *HSIKER clock is selected or*
- *External clock source is configured after setting correctly the define constant EXTERNAL_I2S1_CLOCK_VALUE in the stm32c0xx_hal_conf.h file.*

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_I2S_Transmit()
- Receive an amount of data in blocking mode using HAL_I2S_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_I2S_Transmit_IT()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback
- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_I2S_Receive_IT()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_I2S_Transmit_DMA()
- At transmission end of half transfer HAL_I2S_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxHalfCpltCallback

- At transmission end of transfer HAL_I2S_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_I2S_Receive_DMA()
- At reception end of half transfer HAL_I2S_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxHalfCpltCallback
- At reception end of transfer HAL_I2S_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2S_RxCpltCallback
- In case of transfer Error, HAL_I2S_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2S_ErrorCallback
- Pause the DMA Transfer using HAL_I2S_DMAPause()
- Resume the DMA Transfer using HAL_I2S_DMAResume()
- Stop the DMA Transfer using HAL_I2S_DMAStop() In Slave mode, if HAL_I2S_DMAStop is used to stop the communication, an error HAL_I2S_ERROR_BUSY_LINE_RX is raised as the master continue to transmit data. In this case __HAL_I2S_FLUSH_RX_DR macro must be used to flush the remaining data inside DR register and avoid using DeInit/Init process for the next transfer.

I2S HAL driver macros list

Below the list of most used macros in I2S HAL driver.

- __HAL_I2S_ENABLE: Enable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_DISABLE: Disable the specified SPI peripheral (in I2S mode)
- __HAL_I2S_ENABLE_IT : Enable the specified I2S interrupts
- __HAL_I2S_DISABLE_IT : Disable the specified I2S interrupts
- __HAL_I2S_GET_FLAG: Check whether the specified I2S flag is set or not
- __HAL_I2S_FLUSH_RX_DR: Read DR Register to Flush RX Data

Note: You can refer to the I2S HAL driver header file for more useful macros

I2S HAL driver macros list

Callback registration:

1. The compilation flag USE_HAL_I2S_REGISTER_CALLBACKS when set to 1U allows the user to configure dynamically the driver callbacks. Use Functions HAL_I2S_RegisterCallback() to register an interrupt callback. Function HAL_I2S_RegisterCallback() allows to register following callbacks:
   - TxCpltCallback : I2S Tx Completed callback
   - RxCpltCallback : I2S Rx Completed callback
   - TxHalfCpltCallback : I2S Tx Half Completed callback
   - RxHalfCpltCallback : I2S Rx Half Completed callback
   - ErrorCallback : I2S Error callback
   - MspInitCallback : I2S Msp Init callback
   - MspDeInitCallback : I2S 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_I2S_UnRegisterCallback to reset a callback to the default weak function. HAL_I2S_UnRegisterCallback takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
   - TxCpltCallback : I2S Tx Completed callback
   - RxCpltCallback : I2S Rx Completed callback
   - TxHalfCpltCallback : I2S Tx Half Completed callback
   - RxHalfCpltCallback : I2S Rx Half Completed callback
   - ErrorCallback : I2S Error callback
   - MspInitCallback : I2S Msp Init callback
   - MspDeInitCallback : I2S Msp DeInit callback

By default, after the HAL_I2S_Init() and when the state is HAL_I2S_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_I2S_MasterTxCpltCallback(), HAL_I2S_MasterRxCpltCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_I2S_Init()/ HAL_I2S_DeInit() only when these callbacks are null (not registered beforehand). If MspInit or MspDeInit are not null, the HAL_I2S_Init()/ HAL_I2S_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand) whatever the state.

Callbacks can be registered/unregistered in HAL_I2S_STATE_READY state only. Exception done MspInit/ MspDeInit functions that can be registered/unregistered in HAL_I2S_STATE_READY or HAL_I2S_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_I2S_RegisterCallback() before calling HAL_I2S_DeInit() or HAL_I2S_Init() function.

When the compilation define USE_HAL_I2S_REGISTER_CALLBACKS is set to 0 or not defined, the callback registering feature is not available and weak (surcharged) callbacks are used.

22.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement HAL_I2S_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2S_Init() to configure the selected device with the selected configuration:
  - Mode
  - Standard
  - Data Format
  - MCLK Output
  - Audio frequency
  - Polarity
- Call the function HAL_I2S_DeInit() to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- ***HAL_I2S_Init()***
- ***HAL_I2S_DeInit()***
- ***HAL_I2S_MspInit()***
- ***HAL_I2S_MspDeInit()***
- ***HAL_I2S_RegisterCallback()***
- ***HAL_I2S_UnRegisterCallback()***

22.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S 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 I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
   - HAL_I2S_Transmit()
   - HAL_I2S_Receive()
3. No-Blocking mode functions with Interrupt are :
   - HAL_I2S_Transmit_IT()
   - HAL_I2S_Receive_IT()
4. No-Blocking mode functions with DMA are :
   - HAL_I2S_Transmit_DMA()
   - HAL_I2S_Receive_DMA()

5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
   – HAL_I2S_TxCpltCallback()
   – HAL_I2S_RxCpltCallback()
   – HAL_I2S_ErrorCallback()

This section contains the following APIs:

- ***HAL_I2S_Transmit()***
- ***HAL_I2S_Receive()***
- ***HAL_I2S_Transmit_IT()***
- ***HAL_I2S_Receive_IT()***
- ***HAL_I2S_Transmit_DMA()***
- ***HAL_I2S_Receive_DMA()***
- ***HAL_I2S_DMAPause()***
- ***HAL_I2S_DMAResume()***
- ***HAL_I2S_DMAStop()***
- ***HAL_I2S_IRQHandler()***
- ***HAL_I2S_TxHalfCpltCallback()***
- ***HAL_I2S_TxCpltCallback()***
- ***HAL_I2S_RxHalfCpltCallback()***
- ***HAL_I2S_RxCpltCallback()***
- ***HAL_I2S_ErrorCallback()***

22.2.4 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_I2S_GetState()***
- ***HAL_I2S_GetError()***

22.2.5 Detailed description of functions

HAL_I2S_Init

Function name

HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)

Function description

Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DeInit

Function name

HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)

Function description

DeInitializes the I2S peripheral.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_MspInit

Function name

void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP Init.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_MspDeInit

Function name

void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)

Function description

I2S MSP DeInit.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_RegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID, pI2S_CallbackTypeDef pCallback)

Function description

Register a User I2S Callback To be used instead of the weak predefined callback.

Parameters

- **hi2s:** Pointer to a I2S_HandleTypeDef structure that contains the configuration information for the specified I2S.
- **CallbackID:** ID of the callback to be registered
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_I2S_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_I2S_UnRegisterCallback (I2S_HandleTypeDef * hi2s, HAL_I2S_CallbackIDTypeDef CallbackID)

Function description

Unregister an I2S Callback I2S callback is redirected to the weak predefined callback.

Parameters

- **hi2s:** Pointer to a I2S_HandleTypeDef structure that contains the configuration information for the specified I2S.
- **CallbackID:** ID of the callback to be unregistered

Return values

- **HAL:** status

HAL_I2S_Transmit

Function name

HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Transmit an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 24-bit or 32-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive

Function name

HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)

Function description

Receive an amount of data in blocking mode.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:
- **Timeout:** Timeout duration

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 24-bit or 32-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continuous way and as the I2S is not disabled at the end of the I2S transaction.

HAL_I2S_Transmit_IT

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 24-bit or 32-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_IT

Function name

HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with Interrupt.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 24-bit or 32-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronization between Master and Slave otherwise the I2S interrupt should be optimized.

HAL_I2S_IRQHandler

Function name

void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)

Function description

This function handles I2S interrupt request.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_Transmit_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Transmit an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Transmit data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 24-bit or 32-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_Receive_DMA

Function name

HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)

Function description

Receive an amount of data in non-blocking mode with DMA.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
- **pData:** a 16-bit pointer to the Receive data buffer.
- **Size:** number of data sample to be sent:

Return values

- **HAL:** status

Notes

- When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 24-bit or 32-bit data length.
- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

HAL_I2S_DMAPause

Function name

HAL_StatusTypeDef HAL_I2S_DMAPause (I2S_HandleTypeDef * hi2s)

Function description

Pauses the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAResume

Function name

HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)

Function description

Resumes the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_DMAStop

Function name

HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)

Function description

Stops the audio DMA Stream/Channel playing from the Media.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** status

HAL_I2S_TxHalfCpltCallback

Function name

void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer Half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_TxCpltCallback

Function name

void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Tx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxHalfCpltCallback

Function name

void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_RxCpltCallback

Function name

void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)

Function description

Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_ErrorCallback

Function name

void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)

Function description

I2S error callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **None:**

HAL_I2S_GetState

Function name

HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)

Function description

Return the I2S state.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **HAL:** state

HAL_I2S_GetError

Function name

uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)

Function description

Return the I2S error code.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- **I2S:** Error Code

22.3 I2S Firmware driver defines

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

22.3.1 I2S

I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

I2S Error

HAL_I2S_ERROR_NONE
: No error

HAL_I2S_ERROR_TIMEOUT
: Timeout error

HAL_I2S_ERROR_OVR
: OVR error

HAL_I2S_ERROR_UDR
: UDR error

HAL_I2S_ERROR_DMA
: DMA transfer error

HAL_I2S_ERROR_PRESCALER
: Prescaler Calculation error

HAL_I2S_ERROR_INVALID_CALLBACK
: Invalid Callback error

HAL_I2S_ERROR_BUSY_LINE_RX
: Busy Rx Line error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE

Description:

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE

Description:

- Enable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

Description:

- Disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_ENABLE_IT

Description:

- Enable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_TXE: Tx buffer empty interrupt enable
  - I2S_IT_RXNE: RX buffer not empty interrupt enable
  - I2S_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_I2S_DISABLE_IT

Description:

- Disable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
  - I2S_IT_TXE: Tx buffer empty interrupt enable
  - I2S_IT_RXNE: RX buffer not empty interrupt enable
  - I2S_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_I2S_GET_IT_SOURCE

Description:

- Checks if the specified I2S interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
  - I2S_IT_TXE: Tx buffer empty interrupt enable
  - I2S_IT_RXNE: RX buffer not empty interrupt enable
  - I2S_IT_ERR: Error interrupt enable

Return value:

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

__HAL_I2S_GET_FLAG

Description:

- Checks whether the specified I2S flag is set or not.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - I2S_FLAG_RXNE: Receive buffer not empty flag
  - I2S_FLAG_TXE: Transmit buffer empty flag
  - I2S_FLAG_UDR: Underrun flag
  - I2S_FLAG_OVR: Overrun flag
  - I2S_FLAG_FRE: Frame error flag
  - I2S_FLAG_CHSIDE: Channel Side flag
  - I2S_FLAG_BSY: Busy flag

Return value:

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

__HAL_I2S_CLEAR_OVRFLAG

Description:

- Clears the I2S OVR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_CLEAR_UDRFLAG

Description:

- Clears the I2S UDR pending flag.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_FLUSH_RX_DR

Description:

- Flush the I2S DR Register.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

I2S Flags Definition

I2S_FLAG_TXE

I2S_FLAG_RXNE

I2S_FLAG_UDR

I2S_FLAG_OVR

I2S_FLAG_FRE

I2S_FLAG_CHSIDE

I2S_FLAG_BSY

I2S_FLAG_MASK

I2S Interrupts Definition

I2S_IT_TXE

I2S_IT_RXNE

I2S_IT_ERR

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

23 HAL IRDA Generic Driver

23.1 IRDA Firmware driver registers structures

23.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32c0xx_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***.

23.1.2 __IRDA_HandleTypeDef

__IRDA_HandleTypeDef is defined in the stm32c0xx_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

23.2 IRDA Firmware driver API description

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

23.2.1 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

23.2.2 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.

23.2.3 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()*

23.2.4 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()*

23.2.5 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()*

23.2.6 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.
- When IRDA parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When IRDA parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When IRDA parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When IRDA parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When IRDA parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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).
- When IRDA parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pData.

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

23.3 IRDA Firmware driver defines

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

23.3.1 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

24 HAL IRDA Extension Driver

24.1 IRDAEx Firmware driver defines

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

24.1.1 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

25 HAL IWDG Generic Driver

25.1 IWDG Firmware driver registers structures

25.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32c0xx_hal_iwdg.h

Data Fields

- ***uint32_t Prescaler***
- ***uint32_t Reload***
- ***uint32_t Window***

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

25.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32c0xx_hal_iwdg.h

Data Fields

- ***IWDG_TypeDef * Instance***
- ***IWDG_InitTypeDef Init***

Field Documentation

- ***IWDG_TypeDef* IWDG_HandleTypeDef::Instance***
  Register base address
- ***IWDG_InitTypeDef IWDG_HandleTypeDef::Init***
  IWDG required parameters

25.2 IWDG Firmware driver API description

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

25.2.1 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 / ~32.7s The IWDG timeout may vary due to LSI clock frequency dispersion. STM32C0xx 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).

25.2.2 How to use this driver

1. 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 and IWDG_WINR.
   - 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.
   - Wait for status flags to be reset.
2. 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

25.2.3 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()***

25.2.4 IO operation functions

This section provides functions allowing to:

- Refresh the IWDG.

This section contains the following APIs:

- ***HAL_IWDG_Refresh()***

25.2.5 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_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

25.3 IWDG Firmware driver defines

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

25.3.1 IWDG

IWDG

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 and IWDG_WINR 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 Window option

IWDG_WINDOW_DISABLE

26 HAL PWR Generic Driver

26.1 PWR Firmware driver API description

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

26.1.1 Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_PWR_DeInit()*

26.1.2 Peripheral Control functions

WakeUp pin configuration

- WakeUp pins are used to wakeup the system from Standby mode or Shutdown mode. WakeUp pins polarity can be set to configure event detection on high level (rising edge) or low level (falling edge).

Low Power mode configuration

The devices feature 4 low-power modes:

- Sleep mode: Cortex-M0+ core stopped, peripherals kept running, regulator is main mode.
- Stop 0 mode: all clocks are stopped except LSI and LSE, regulator is main mode.
- Standby mode: all clocks are stopped except LSI and LSE, regulator is disable.
- Shutdown mode: all clocks are stopped except LSE, regulator is disable.

Sleep mode

- Entry: The Sleep is entered through HAL_PWR_EnterSLEEPMode() API specifying the main regulator per default and if exit is interrupt or event triggered.
  - PWR_MAINREGULATOR_ON: Sleep mode (regulator in main mode).
  - PWR_SLEEPENTRY_WFI: Core enters sleep mode with WFI instruction
  - PWR_SLEEPENTRY_WFE: Core enters sleep mode with WFE instruction
- WFI Exit:
  - Any interrupt enabled in nested vectored interrupt controller (NVIC)
- WFE Exit:
  - Any wakeup event if cortex is configured with SEVONPEND = 0
  - Interrupt even when disabled in NVIC if cortex is configured with SEVONPEND = 1

Stop 0 mode

- Entry: The Stop modes are entered through the following APIs:
  - HAL_PWR_EnterSTOPMode() with following settings:
    - PWR_MAINREGULATOR_ON to enter STOP0 mode.
- Exit (interrupt or event-triggered, specified when entering STOP mode):
  - PWR_STOPENTRY_WFI: enter Stop mode with WFI instruction
  - PWR_STOPENTRY_WFE: enter Stop mode with WFE instruction
- WFI Exit:
  - Any EXTI line (internal or external) configured in interrupt mode with corresponding interrupt enable in NVIC

- WFE Exit:
  - Any EXTI line (internal or external) configured in event mode if cortex is configured with SEVONPEND = 0
  - Any EXTI line configured in interrupt mode (even if the corresponding EXTI Interrupt vector is disabled in the NVIC) if cortex is configured with SEVONPEND = 0. The interrupt source can be external interrupts or peripherals with wakeup capability.

Standby mode

- Entry:
  - The Standby mode is entered through HAL_PWR_EnterSTANDBYMode() API, by setting SLEEPDEEP in Cortex control register.
- Exit:
  - WKUP pin edge detection, RTC event (alarm, timestamp), LSE CSS detection, reset on NRST pin, IWDG reset & BOR reset.

Exiting Standby generates a power reset: Cortex is reset and execute Reset handler vector, all registers in the Vcore domain are set to their reset value. Registers outside the VCORE domain (RTC, WKUP, IWDG, and Standby/Shutdown modes control) are not impacted.

Shutdown mode

In Shutdown mode, voltage regulator is disabled, all clocks are off except LSE, RRS bit is cleared. SRAM and registers contents are lost except for backup domain registers.

- Entry:
  - The Shutdown mode is entered through HAL_PWREx_EnterSHUTDOWNMode() API, by setting SLEEPDEEP in Cortex control register.
- Exit:
  - WKUP pin edge detection, RTC event (alarm, timestamp), LSE CSS detection, reset on NRST pin.

Exiting Shutdown generates a brown out reset: Cortex is reset and execute Reset handler vector, all registers are set to their reset value but ones in backup domain.

This section contains the following APIs:

- ***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()***

26.1.3 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:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinPolarity)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPinPolarity:** Specifies which Wake-Up pin to enable. This parameter can be one of the following legacy values which set the default polarity i.e. detection on high level (rising edge):
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5 (Only on STM32C071xx), PWR_WAKEUP_PIN6 or one of the following value where the user can explicitly specify the enabled pin and the chosen polarity:
  - PWR_WAKEUP_PIN1_HIGH or PWR_WAKEUP_PIN1_LOW
  - PWR_WAKEUP_PIN2_HIGH or PWR_WAKEUP_PIN2_LOW
  - PWR_WAKEUP_PIN3_HIGH or PWR_WAKEUP_PIN3_LOW
  - PWR_WAKEUP_PIN4_HIGH or PWR_WAKEUP_PIN4_LOW
  - PWR_WAKEUP_PIN5_HIGH or PWR_WAKEUP_PIN5_LOW (*)
  - PWR_WAKEUP_PIN6_HIGH or PWR_WAKEUP_PIN6_LOW

Return values

- **None:**

Notes

- PWR_WAKEUP_PINx and PWR_WAKEUP_PINx_HIGH are equivalent.
- (*) Availability depends on devices

HAL_PWR_DisableWakeUpPin

Function name

void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)

Function description

Disable the WakeUp PINx functionality.

Parameters

- **WakeUpPinx:** Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values:
  - PWR_WAKEUP_PIN1, PWR_WAKEUP_PIN2, PWR_WAKEUP_PIN3, PWR_WAKEUP_PIN4, PWR_WAKEUP_PIN5 (*), PWR_WAKEUP_PIN6

Return values

- **None:**

Notes

- (*) Availability depends on devices

HAL_PWR_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)

Function description

Enter Sleep mode.

Parameters

- **Regulator:** Specifies the regulator state in Sleep mode. This parameter can be the following value:
  - PWR_MAINREGULATOR_ON Sleep mode (regulator in main mode)
- **SLEEPEntry:** Specifies if Sleep mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_SLEEPENTRY_WFI enter Sleep or Low-power Sleep mode with WFI instruction
  - PWR_SLEEPENTRY_WFE enter Sleep or Low-power Sleep mode with WFE instruction

Return values

- **None:**

Notes

- In Sleep mode, all I/O pins keep the same state as in Run mode.
- When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source.

HAL_PWR_EnterSTOPMode

Function name

void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)

Function description

Enter Stop mode.

Parameters

- **Regulator:** Specifies the regulator state in Stop mode This parameter can be of the following value:
  - PWR_MAINREGULATOR_ON Stop 0 mode (main regulator ON)
- **STOPEntry:** Specifies Stop 0 mode is entered with WFI or WFE instruction. This parameter can be one of the following values:
  - PWR_STOPENTRY_WFI Enter Stop 0 mode with WFI instruction.
  - PWR_STOPENTRY_WFE Enter Stop 0 mode with WFE instruction.

Return values

- **None:**

Notes

- This API is named HAL_PWR_EnterSTOPMode to ensure compatibility with legacy code running on devices where only "Stop mode" is mentioned with main regulator ON.
- In Stop mode, all I/O pins keep the same state as in Run mode.
- All clocks in the VCORE domain are stopped; the HSI and the HSE oscillators are disabled. Some peripherals with the wakeup capability can switch on the HSI to receive a frame, and switch off the HSI after receiving the frame if it is not a wakeup frame. SRAM and register contents are preserved.
- When exiting Stop 0 mode by issuing an interrupt or a wakeup event, the HSI RC oscillator is selected as system clock

HAL_PWR_EnterSTANDBYMode

Function name

void HAL_PWR_EnterSTANDBYMode (void)

Function description

Enter Standby mode.

Return values

- **None:**

Notes

- In Standby mode,the HSI and the HSE oscillators are switched off. SRAM and register contents are lost except for registers in the Backup domain and Standby circuitry.
- The I/Os can be configured either with a pull-up or pull-down or can be kept in analog state. HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() respectively enable Pull Up and PullDown state. HAL_PWREx_DisableGPIOPullUp() & HAL_PWREx_DisableGPIOPullDown() disable the same. These states are effective in Standby mode only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Enable Sleep-On-Exit Cortex feature.

Return values

- **None:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor enters SLEEP or DEEPSLEEP mode when an interruption handling is over returning to thread mode. 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 Cortex feature.

Return values

- **None:**

Notes

- Clear SLEEPONEXIT bit of SCR register. When this bit is set, the processor enters SLEEP or DEEPSLEEP mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable Cortex Sev On Pending feature.

Return values

- **None:**

Notes

- Set SEVONPEND bit of SCR register. When this bit is set, enabled events and all interrupts, including disabled ones can wakeup processor from WFE.

HAL_PWR_DisableSEVOnPend

Function name

void HAL_PWR_DisableSEVOnPend (void)

Function description

Disable Cortex Sev On Pending feature.

Return values

- **None:**

Notes

- Clear SEVONPEND bit of SCR register. When this bit is clear, only enable interrupts or events can wakeup processor from WFE

26.2 PWR Firmware driver defines

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

26.2.1 PWR

PWR

PWR Exported Macros

__HAL_PWR_GET_FLAG

Description:

- Check whether or not a specific PWR flag is set.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one a combination of following values:
  - PWR_FLAG_WUF1: Wake Up Flag 1. Indicates that a wakeup event was received from the WKUP pin 1.
  - PWR_FLAG_WUF2: Wake Up Flag 2. Indicates that a wakeup event was received from the WKUP pin 2.
  - PWR_FLAG_WUF3: Wake Up Flag 3. Indicates that a wakeup event was received from the WKUP pin 3.
  - PWR_FLAG_WUF4: Wake Up Flag 4. Indicates that a wakeup event was received from the WKUP pin 4.
  - PWR_FLAG_WUF5: Wake Up Flag 5. Indicates that a wakeup event was received from the WKUP pin 5. (*)
  - PWR_FLAG_WUF6: Wake Up Flag 6. Indicates that a wakeup event was received from the WKUP pin 6.
  - PWR_FLAG_SB: StandBy Flag. Indicates that the system entered StandBy mode.
  - PWR_FLAG_WUFI: Wake-Up Flag Internal. Set when a wakeup is detected on the internal wakeup line. OR a combination of following values:
  - PWR_FLAG_FLASH_READY: Flash is ready. Indicates whether flash can be used or not
  - PWR_FLAG_PVMO_USB: Peripheral Voltage Monitoring Output. Indicates whether VDDUSB voltage is below or above PVM threshold. (*)

Return value:

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

Notes:

- (*) Availability depends on devices

__HAL_PWR_CLEAR_FLAG

Description:

- Clear a specific PWR flag.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be a combination of following values:
  - PWR_FLAG_WUF1: Wake Up Flag 1. Indicates that a wakeup event was received from the WKUP pin 1.
  - PWR_FLAG_WUF2: Wake Up Flag 2. Indicates that a wakeup event was received from the WKUP pin 2.
  - PWR_FLAG_WUF3: Wake Up Flag 3. Indicates that a wakeup event was received from the WKUP pin 3.
  - PWR_FLAG_WUF4: Wake Up Flag 4. Indicates that a wakeup event was received from the WKUP pin 4.
  - PWR_FLAG_WUF5: Wake Up Flag 5. Indicates that a wakeup event was received from the WKUP pin 5. (*)
  - PWR_FLAG_WUF6: Wake Up Flag 6. Indicates that a wakeup event was received from the WKUP pin 6.
  - PWR_FLAG_SB: Standby Flag. Indicates that the system entered Standby mode.

Return value:

- None

Notes:

- (*) Availability depends on devices

PWR Low Power Mode Selection

PWR_LOWPOWERMODE_STOP0

Stop 0: stop mode with main regulator

PWR_LOWPOWERMODE_STANDBY

Standby mode

PWR_LOWPOWERMODE_SHUTDOWN

Shutdown mode

PWR regulator mode

PWR_MAINREGULATOR_ON

Regulator in main 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 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 WakeUp pins

PWR_WAKEUP_PIN1

Wakeup pin 1 (with high level detection)

PWR_WAKEUP_PIN2

Wakeup pin 2 (with high level detection)

PWR_WAKEUP_PIN3

Wakeup pin 3 (with high level detection)

PWR_WAKEUP_PIN4

Wakeup pin 4 (with high level detection)

PWR_WAKEUP_PIN5

Wakeup pin 5 (with high level detection)

PWR_WAKEUP_PIN6

Wakeup pin 6 (with high level detection)

PWR_WAKEUP_PIN1_HIGH

Wakeup pin 1 (with high level detection)

PWR_WAKEUP_PIN2_HIGH

Wakeup pin 2 (with high level detection)

PWR_WAKEUP_PIN3_HIGH

Wakeup pin 3 (with high level detection)

PWR_WAKEUP_PIN4_HIGH

Wakeup pin 4 (with high level detection)

PWR_WAKEUP_PIN5_HIGH

Wakeup pin 5 (with high level detection)

PWR_WAKEUP_PIN6_HIGH

Wakeup pin 6 (with high level detection)

PWR_WAKEUP_PIN1_LOW

Wakeup pin 1 (with low level detection)

PWR_WAKEUP_PIN2_LOW

Wakeup pin 2 (with low level detection)

PWR_WAKEUP_PIN3_LOW

Wakeup pin 3 (with low level detection)

PWR_WAKEUP_PIN4_LOW

Wakeup pin 4 (with low level detection)

PWR_WAKEUP_PIN5_LOW

Wakeup pin 5 (with low level detection)

PWR_WAKEUP_PIN6_LOW

Wakeup pin 6 (with low level detection)

27 HAL PWR Extension Driver

27.1 PWREx Firmware driver API description

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

27.1.1 Extended Peripheral Initialization and de-initialization functions

This section contains the following APIs:

- *HAL_PWREx_EnableInternalWakeUpLine()*
- *HAL_PWREx_DisableInternalWakeUpLine()*
- *HAL_PWREx_EnableGPIOPullUp()*
- *HAL_PWREx_DisableGPIOPullUp()*
- *HAL_PWREx_EnableGPIOPullDown()*
- *HAL_PWREx_DisableGPIOPullDown()*
- *HAL_PWREx_EnablePullUpPullDownConfig()*
- *HAL_PWREx_DisablePullUpPullDownConfig()*
- *HAL_PWREx_EnableFlashPowerDown()*
- *HAL_PWREx_DisableFlashPowerDown()*
- *HAL_PWREx_EnterSHUTDOWNMode()*
- *HAL_PWREx_BKUPWrite()*
- *HAL_PWREx_BKUPRead()*

27.1.2 Extended PWR Backup register functions

This subsection provides functions allowing to

- Write a data in a specified PWR Backup data register
- Read a data in a specified PWR Backup data register

This section contains the following APIs:

- *HAL_PWREx_BKUPWrite()*
- *HAL_PWREx_BKUPRead()*

27.1.3 Detailed description of functions

HAL_PWREx_EnableInternalWakeUpLine

Function name

void HAL_PWREx_EnableInternalWakeUpLine (void)

Function description

Enable Internal Wake-up Line.

Return values

- **None:**

HAL_PWREx_DisableInternalWakeUpLine

Function name

void HAL_PWREx_DisableInternalWakeUpLine (void)

Function description

Disable Internal Wake-up Line.

Return values

- **None:**

HAL_PWREx_EnableGPIOPullUp

Function name

HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO:** Specify the IO port. This parameter can be PWR_GPIO_A, ..., PWR_GPIO_F to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for ports where less I/O pins are available) or the logical OR of several of them to set several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Set the relevant PUy bit 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.

HAL_PWREx_DisableGPIOPullUp

Function name

HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-up state in Standby mode and Shutdown modes.

Parameters

- **GPIO:** Specifies the IO port. This parameter can be PWR_GPIO_A, ..., PWR_GPIO_F to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for ports where less I/O pins are available) or the logical OR of several of them to reset several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Reset the relevant PUy bit of PWR_PUCRx register used to configure the I/O in pull-up state in Standby and Shutdown modes.

HAL_PWREx_EnableGPIOPullDown

Function name

HAL_StatusTypeDef HAL_PWREx_EnableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** Specify the IO port. This parameter can be PWR_GPIO_A..PWR_GPIO_F to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for ports where less I/O pins are available) or the logical OR of several of them to set several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Set the relevant PDy bit 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.

HAL_PWREx_DisableGPIOPullDown

Function name

HAL_StatusTypeDef HAL_PWREx_DisableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** Specifies the IO port. This parameter can be PWR_GPIO_A..PWR_GPIO_F to select the GPIO peripheral.
- **GPIONumber:** Specify the I/O pins numbers. This parameter can be one of the following values: PWR_GPIO_BIT_0, ..., PWR_GPIO_BIT_15 (except for ports where less I/O pins are available) or the logical OR of several of them to reset several bits for a given port in a single API call.

Return values

- **HAL:** Status

Notes

- Reset the relevant PDy bit of PWR_PDCRx register used to configure the I/O in pull-down state in Standby and Shutdown modes.

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() APIs 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_EnableFlashPowerDown

Function name

void HAL_PWREx_EnableFlashPowerDown (uint32_t PowerMode)

Function description

Enable Flash Power Down.

Parameters

- **PowerMode:** this can be a combination of following values:
  - PWR_FLASHPD_SLEEP
  - PWR_FLASHPD_STOP

Return values

- **None:**

Notes

- This API allows to enable flash power down capabilities in sleep and stop modes.

HAL_PWREx_DisableFlashPowerDown

Function name

void HAL_PWREx_DisableFlashPowerDown (uint32_t PowerMode)

Function description

Disable Flash Power Down.

Parameters

- **PowerMode:** this can be a combination of following values:
  - PWR_FLASHPD_SLEEP
  - PWR_FLASHPD_STOP

Return values

- **None:**

Notes

- This API allows to disable flash power down capabilities in sleep and stop modes.

HAL_PWREx_BKUPWrite

Function name

void HAL_PWREx_BKUPWrite (uint32_t BackupRegister, uint16_t Data)

Function description

Write a data in a specified PWR Backup data register.

Parameters

- **BackupRegister:** PWR Backup data Register number. This parameter can be PWR_BKP_DRx where x can be from 0 to PWR_BACKUP_NB
- **Data:** Data to be written in the specified Backup data register.

Return values

- **None:**

HAL_PWREx_BKUPRead

Function name

uint32_t HAL_PWREx_BKUPRead (uint32_t BackupRegister)

Function description

Reads data from the specified PWR Backup data Register.

Parameters

- **BackupRegister:** PWR Backup data Register number. This parameter can be PWR_BKP_DRx where x can be from 0 to PWR_BACKUP_NB

Return values

- **Read:** value

HAL_PWREx_EnterSHUTDOWNMode

Function name

void HAL_PWREx_EnterSHUTDOWNMode (void)

Function description

Enter Shutdown mode.

Return values

- **None:**
- **None:**

Notes

- In Shutdown mode, the PLL, the HSI, the LSI and the HSE oscillators are switched off. The voltage regulator is disabled and Vcore domain is powered off. SRAM and registers contents are lost except for registers in the Backup domain. The BOR is not available.
- The I/Os can be configured either with a pull-up or pull-down or can be kept in analog state. HAL_PWREx_EnableGPIOPullUp() and HAL_PWREx_EnableGPIOPullDown() respectively enable Pull Up and PullDown state. HAL_PWREx_DisableGPIOPullUp() & HAL_PWREx_DisableGPIOPullDown() disable the same. These states are effective in Standby mode only if APC bit is set through HAL_PWREx_EnablePullUpPullDownConfig() API.

27.2 PWREx Firmware driver defines

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

27.2.1 PWREx

PWREx

PWREx Backup Registers Definition

PWR_BKP_NUMBER

PWR_BKP_DR0

PWR_BKP_DR1

PWR_BKP_DR2

PWR_BKP_DR3

PWR Status Flags

PWR_FLAG_WUF1

Wakeup event on wakeup pin 1

PWR_FLAG_WUF2

Wakeup event on wakeup pin 2

PWR_FLAG_WUF3

Wakeup event on wakeup pin 3

PWR_FLAG_WUF4

Wakeup event on wakeup pin 4

PWR_FLAG_WUF5

Wakeup event on wakeup pin 5

PWR_FLAG_WUF6

Wakeup event on wakeup pin 6

PWR_FLAG_WUF

Wakeup event on all wakeup pin

PWR_FLAG_SB

Standby flag

PWR_FLAG_WUFI

Wakeup on internal wakeup line

PWR_FLAG_FLASH_READY

Flash ready

Flash Power Down modes

PWR_FLASHPD_SLEEP

Enable Flash power down in sleep mode

PWR_FLASHPD_STOP

Enable Flash power down in stop mode

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

PWREx Gpio Pin Number

PWR_GPIO_PIN_NB
Number of gpio pin in bank

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_F

GPIO port F

PWR PVM Mode Mask

PVM_MODE_IT

Mask for interruption yielded by PVM threshold crossing

PVM_MODE_EVT

Mask for event yielded by PVM threshold crossing

PVM_RISING_EDGE

Mask for rising edge set as PVM trigger

PVM_FALLING_EDGE

Mask for falling edge set as PVM trigger

Shift to apply to retrieve polarity information from PWR_WAKEUP_PINy_xxx constants

PWR_WUP_POLARITY_SHIFT

Internal constant used to retrieve wakeup pin polariry

28 HAL RCC Generic Driver

28.1 RCC Firmware driver registers structures

28.1.1 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32c0xx_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t LSEState***
- ***uint32_t HSIState***
- ***uint32_t HSIDiv***
- ***uint32_t HSICalibrationValue***
- ***uint32_t LSIState***

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::HSIDiv***
  The division factor of the HSI48. This parameter can be a value of ***RCC_HSI_Div***
- ***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
- ***uint32_t RCC_OscInitTypeDef::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***

28.1.2 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32c0xx_hal_rcc.h

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t SYSCLKDivider***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
  The clock to be configured. This parameter can be a combination 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::SYSCLKDivider***
  The system clock divider. This parameter can be a value of ***RCC_SYS_Clock_Divider***
- ***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_HCLK_Clock_Divider***

- ***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_Clock_Divider***

28.2 RCC Firmware driver API description

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

28.2.1 RCC specific features

After reset the device is running from High Speed Internal oscillator divided by 4 that is the default system clock with Flash 0 wait state.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses: all peripherals mapped on these busses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in analog mode

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 (RTC, ADC, USART, I2C, I2S)

28.2.2 Initialization and de-initialization functions

This section provides functions allowing to configure the internal and external oscillators (HSE, HSI, LSE, LSI, CSS and MCO) and the System busses clocks (SYSCLK, AHB, APB)

Internal/external clock

- HSI (high-speed internal): 48 MHz factory-trimmed RC used directly 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.
- CSS (Clock security system): once enabled, if a HSE or LSE clock failure occurs, the System clock is automatically switched respectively to HSI or LSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M0+ NMI (Non-Maskable Interrupt) exception vector.
- MCOx (microcontroller clock output): used to output LSI, HSI, LSE, HSE and SYSCLK on different pins.

System, AHB and APB busses clocks configuration

- Several clock sources can be used to drive the System clock (SYSCLK): HSI, HSE, LSI and LSE. 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...).and APB (PCLK1) clock is derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use "@ref HAL_RCC_GetSysClockFreq()" function to retrieve the frequencies of these clocks.

Note: *All the peripheral clocks are derived from the System clock (SYSCLK) except:*

- *RTC: the RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 31. You have to use @ref __HAL_RCC_RTC_ENABLE() and @ref HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*
- *ADC: the ADC clock can be derived either from system clock or HSI*
- *USART: the USART clock can be derived either from APB clock or HSI or LSE*
- *I2C: the I2C clock can be derived either from system clock or HSI*
- *I2S: the I2S clock can be derived either from system clock or HSI or external clock source*

- The maximum frequency of the SYSCLK, HCLK, PCLK is 48 MHz.

This section contains the following APIs:

- *HAL_RCC_DeInit()*
- *HAL_RCC_OscConfig()*
- *HAL_RCC_ClockConfig()*

28.2.3 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_GetOscConfig()*
- *HAL_RCC_GetClockConfig()*
- *HAL_RCC_EnableCSS()*
- *HAL_RCC_EnableLSECSS()*
- *HAL_RCC_DisableLSECSS()*
- *HAL_RCC_NMI_IRQHandler()*
- *HAL_RCC_CSSCallback()*
- *HAL_RCC_LSECSSCallback()*

28.2.4 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: HSI ON and used as system clock sourceHSE OFFSYSDIV, AHB and APB prescaler set to 1.CSS, MCO1, MCO2 OFFAll interrupts disabled
- This function does not modify the configuration of the Peripheral clocksLSI, LSE and RTC clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (const RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Initialize the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

Parameters

- **RCC_OscInitStruct:** pointer to a RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.

Return values

- **HAL:** status

Notes

- 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 to HSE On or HSE Bypass.
- Transition 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 to LSE On or LSE Bypass.

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (const RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)

Function description

Initialize the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.

Parameters

- **RCC_ClkInitStruct:** pointer to a RCC_ClkInitTypeDef 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

Return values

- **None:**

Notes

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated by HAL_RCC_GetHCLKFreq() function called within this function
- The HSI is used by default as system clock source after startup from Reset, wake-up from STANDBY mode. After restart from Reset, the HSI frequency is set to 12 Mhz, as HSI divider is set to 4.
- The HSI can be selected as system clock source after 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).
- The LSI can be selected as system clock source after in case of failure of the LSE used directly or indirectly as system clock (if the Clock Security System LSECSS is enabled).
- A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay). 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 MCOx pin.

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source.
  - RCC_MCO1 Clock source to output on MCO1 pins.
  - RCC_MCO2 Clock source to output on MCO2 pins.
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCO1SOURCE_NOCLOCK MCO1 output disabled, no clock on MCO1
  - RCC_MCO1SOURCE_SYSCLK system clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI HSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_HSI48 HSI48 clock selected as MCO1 source (*)
  - RCC_MCO1SOURCE_HSE HSE clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSI LSI clock selected as MCO1 source
  - RCC_MCO1SOURCE_LSE LSE clock selected as MCO1 source
  - RCC_MCO2SOURCE_NOCLOCK MCO2 output disabled, no clock on MCO2
  - RCC_MCO2SOURCE_SYSCLK system clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSI HSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_HSI48 HSI48 clock selected as MCO2 source (*)
  - RCC_MCO2SOURCE_HSE HSE clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSI LSI clock selected as MCO2 source
  - RCC_MCO2SOURCE_LSE LSE clock selected as MCO2 source
- **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
  - RCC_MCODIV_32 division by 32 applied to MCO clock
  - RCC_MCODIV_64 division by 64 applied to MCO clock
  - RCC_MCODIV_128 division by 128 applied to MCO clock

Return values

- **None:**

Notes

- PA8 for MCO1 and PB2 for MCO2 should be configured in alternate function mode.
- (*) peripheral not available on all devices

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-M0+ NMI (Non-Maskable Interrupt) exception vector.
- The Clock Security System can only be cleared by reset.

HAL_RCC_EnableLSECSS

Function name

void HAL_RCC_EnableLSECSS (void)

Function description

Enable the LSE Clock Security System.

Return values

- **None:**

Notes

- If a failure is detected on the LSE 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-M0+ NMI (Non-Maskable Interrupt) exception vector.
- The LSE Clock Security System Detection bit (LSECSSD in CSR1) can only be cleared by a backup domain reset.

HAL_RCC_DisableLSECSS

Function name

void HAL_RCC_DisableLSECSS (void)

Function description

Disable the LSE Clock Security System.

Return values

- **None:**

Notes

- After LSE failure detection, the software must disable LSECSSON
- The Clock Security System can only be cleared by reset otherwise.

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 HSI, function returns values based on HSI_VALUE/HSIDIV(*)
- If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)
- If SYSCLK source is LSI, function returns values based on LSI_VALUE(***)
- If SYSCLK source is LSE, function returns values based on LSE_VALUE(****)
- (*) HSI_VALUE is a constant defined in stm32c0xx_hal_conf.h file (default value 48 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (**) HSE_VALUE is a constant defined in stm32c0xx_hal_conf.h file (default value 48 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.
- (***) LSE_VALUE is a constant defined in stm32c0xx_hal_conf.h file (default value 32768 Hz).
- (****) LSI_VALUE is a constant defined in stm32c0xx_hal_conf.h file (default value 32000 Hz).
- 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_GetOscConfig

Function name

void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

Function description

Configure the RCC_OscInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_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 * RCC_ClkInitStruct, uint32_t * pFLatency)

Function description

Configure the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **RCC_ClkInitStruct:** Pointer to a RCC_ClkInitTypeDef structure that will be configured.
- **pFLatency:** Pointer on the Flash Latency.

Return values

- **None:**

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

Handle the RCC HSE Clock Security System interrupt callback.

Return values

- **none:**

HAL_RCC_LSECSSCallback

Function name

void HAL_RCC_LSECSSCallback (void)

Function description

RCC LSE Clock Security System interrupt callback.

Return values

- **none:**

28.3 RCC Firmware driver defines

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

28.3.1 RCC

RCC

AHB Peripheral Clock Sleep Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SRAM_IS_CLK_SLEEP_ENABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DMA1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FLASH_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SRAM_IS_CLK_SLEEP_DISABLED

__HAL_RCC_CRC_IS_CLK_SLEEP_DISABLED

AHB Peripherals Clock Sleep Enable Disable

__HAL_RCC_DMA1_CLK_SLEEP_ENABLE

__HAL_RCC_FLASH_CLK_SLEEP_ENABLE

__HAL_RCC_SRAM_CLK_SLEEP_ENABLE

__HAL_RCC_CRC_CLK_SLEEP_ENABLE

__HAL_RCC_DMA1_CLK_SLEEP_DISABLE

__HAL_RCC_FLASH_CLK_SLEEP_DISABLE

__HAL_RCC_SRAM_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE

AHB Peripheral Force Release Reset

__HAL_RCC_AHB_FORCE_RESET

__HAL_RCC_DMA1_FORCE_RESET

__HAL_RCC_FLASH_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_AHB_RELEASE_RESET

__HAL_RCC_DMA1_RELEASE_RESET

__HAL_RCC_FLASH_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

AHB Peripheral Clock Enabled or Disabled Status

__HAL_RCC_DMA1_IS_CLK_ENABLED

__HAL_RCC_FLASH_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_DMA1_IS_CLK_DISABLED

__HAL_RCC_FLASH_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

AHB Peripheral Clock Enable Disable

__HAL_RCC_DMA1_CLK_ENABLE

__HAL_RCC_FLASH_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_DMA1_CLK_DISABLE

__HAL_RCC_FLASH_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

RCC APB1 Clock Divider

RCC_APB1_DIV1
: APB not divided

RCC_APB1_DIV2
: APB divided by 2

RCC_APB1_DIV4
: APB divided by 4

RCC_APB1_DIV8
: APB divided by 4

RCC_APB1_DIV16
: APB divided by 16

APB1_GRP1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_TIM3_IS_CLK_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_ENABLED

__HAL_RCC_WWDG_IS_CLK_ENABLED

__HAL_RCC_FDCAN1_IS_CLK_ENABLED

__HAL_RCC_SPI2_IS_CLK_ENABLED

__HAL_RCC_USART2_IS_CLK_ENABLED

__HAL_RCC_USART3_IS_CLK_ENABLED

__HAL_RCC_USART4_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_I2C2_IS_CLK_ENABLED

__HAL_RCC_DBGMCU_IS_CLK_ENABLED

__HAL_RCC_PWR_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_DISABLED

__HAL_RCC_WWDG_IS_CLK_DISABLED

__HAL_RCC_FDCAN1_IS_CLK_DISABLED

__HAL_RCC_SPI2_IS_CLK_DISABLED

__HAL_RCC_USART2_IS_CLK_DISABLED

__HAL_RCC_USART3_IS_CLK_DISABLED

__HAL_RCC_USART4_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

__HAL_RCC_I2C2_IS_CLK_DISABLED

__HAL_RCC_DBGMCU_IS_CLK_DISABLED

__HAL_RCC_PWR_IS_CLK_DISABLED

APB1_GRP1 Peripheral Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_TIM3_CLK_ENABLE

__HAL_RCC_RTCAPB_CLK_ENABLE

__HAL_RCC_WWDG_CLK_ENABLE

__HAL_RCC_FDCAN1_CLK_ENABLE

__HAL_RCC_SPI2_CLK_ENABLE

__HAL_RCC_USART2_CLK_ENABLE

__HAL_RCC_USART3_CLK_ENABLE

__HAL_RCC_USART4_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_I2C2_CLK_ENABLE

__HAL_RCC_DBGMCU_CLK_ENABLE

__HAL_RCC_PWR_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_TIM3_CLK_DISABLE

__HAL_RCC_RTCAPB_CLK_DISABLE

__HAL_RCC_WWDG_CLK_DISABLE

__HAL_RCC_FDCAN1_CLK_DISABLE

__HAL_RCC_SPI2_CLK_DISABLE

__HAL_RCC_USART2_CLK_DISABLE

__HAL_RCC_USART3_CLK_DISABLE

__HAL_RCC_USART4_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

__HAL_RCC_I2C2_CLK_DISABLE

__HAL_RCC_DBGMCU_CLK_DISABLE

__HAL_RCC_PWR_CLK_DISABLE

APB1_GRP1 Peripheral Clock Sleep Enabled or

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_FDCAN1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART4_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_DBGMCU_IS_CLK_SLEEP_ENABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_RTCAPB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_WWDG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_FDCAN1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART4_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_I2C2_IS_CLK_SLEEP_DISABLED

__HAL_RCC_DBGMCU_IS_CLK_SLEEP_DISABLED

__HAL_RCC_PWR_IS_CLK_SLEEP_DISABLED

APB1_GRP1 Peripheral Clock Sleep Enable Disable

__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_ENABLE

__HAL_RCC_WWDG_CLK_SLEEP_ENABLE

__HAL_RCC_FDCAN1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI2_CLK_SLEEP_ENABLE

__HAL_RCC_USART2_CLK_SLEEP_ENABLE

__HAL_RCC_USART3_CLK_SLEEP_ENABLE

__HAL_RCC_USART4_CLK_SLEEP_ENABLE

__HAL_RCC_I2C1_CLK_SLEEP_ENABLE

__HAL_RCC_I2C2_CLK_SLEEP_ENABLE

__HAL_RCC_DBGMCU_CLK_SLEEP_ENABLE

__HAL_RCC_PWR_CLK_SLEEP_ENABLE

__HAL_RCC_TIM2_CLK_SLEEP_DISABLE

__HAL_RCC_TIM3_CLK_SLEEP_DISABLE

__HAL_RCC_RTCAPB_CLK_SLEEP_DISABLE

__HAL_RCC_WWDG_CLK_SLEEP_DISABLE

__HAL_RCC_FDCAN1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI2_CLK_SLEEP_DISABLE

__HAL_RCC_USART2_CLK_SLEEP_DISABLE

__HAL_RCC_USART3_CLK_SLEEP_DISABLE

__HAL_RCC_USART4_CLK_SLEEP_DISABLE

__HAL_RCC_I2C1_CLK_SLEEP_DISABLE

__HAL_RCC_I2C2_CLK_SLEEP_DISABLE

__HAL_RCC_DBGMCU_CLK_SLEEP_DISABLE

__HAL_RCC_PWR_CLK_SLEEP_DISABLE

APB1_GRP1 Peripheral Force Release Reset

__HAL_RCC_APB1_GRP1_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_TIM3_FORCE_RESET

__HAL_RCC_FDCAN1_FORCE_RESET

__HAL_RCC_SPI2_FORCE_RESET

__HAL_RCC_USART2_FORCE_RESET

__HAL_RCC_USART3_FORCE_RESET

__HAL_RCC_USART4_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_I2C2_FORCE_RESET

__HAL_RCC_DBGMCU_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET

__HAL_RCC_APB1_GRP1_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_TIM3_RELEASE_RESET

__HAL_RCC_FDCAN1_RELEASE_RESET

__HAL_RCC_SPI2_RELEASE_RESET

__HAL_RCC_USART2_RELEASE_RESET

__HAL_RCC_USART3_RELEASE_RESET

__HAL_RCC_USART4_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

__HAL_RCC_I2C2_RELEASE_RESET

__HAL_RCC_DBGMCU_RELEASE_RESET

__HAL_RCC_PWR_RELEASE_RESET

APB1_GRP2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_TIM1_IS_CLK_ENABLED

__HAL_RCC_SPI1_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_TIM14_IS_CLK_ENABLED

__HAL_RCC_TIM15_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_ADC_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_TIM1_IS_CLK_DISABLED

__HAL_RCC_SPI1_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_TIM14_IS_CLK_DISABLED

__HAL_RCC_TIM15_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

__HAL_RCC_ADC_IS_CLK_DISABLED

APB1_GRP2 Peripheral Clock Enable Disable

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_TIM1_CLK_ENABLE

__HAL_RCC_SPI1_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_TIM14_CLK_ENABLE

__HAL_RCC_TIM15_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_ADC_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_TIM1_CLK_DISABLE

__HAL_RCC_SPI1_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_TIM14_CLK_DISABLE

__HAL_RCC_TIM15_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

__HAL_RCC_ADC_CLK_DISABLE

APB1_GRP2 Peripheral Clock Sleep Enabled or

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM14_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_ENABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_ENABLED

__HAL_RCC_ADC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SPI1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM14_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM15_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM16_IS_CLK_SLEEP_DISABLED

__HAL_RCC_TIM17_IS_CLK_SLEEP_DISABLED

__HAL_RCC_ADC_IS_CLK_SLEEP_DISABLED

APB1_GRP2 Peripheral Clock Sleep Enable Disable

__HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

__HAL_RCC_TIM1_CLK_SLEEP_ENABLE

__HAL_RCC_SPI1_CLK_SLEEP_ENABLE

__HAL_RCC_USART1_CLK_SLEEP_ENABLE

__HAL_RCC_TIM14_CLK_SLEEP_ENABLE

__HAL_RCC_TIM15_CLK_SLEEP_ENABLE

__HAL_RCC_TIM16_CLK_SLEEP_ENABLE

__HAL_RCC_TIM17_CLK_SLEEP_ENABLE

__HAL_RCC_ADC_CLK_SLEEP_ENABLE

__HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE

__HAL_RCC_TIM1_CLK_SLEEP_DISABLE

__HAL_RCC_SPI1_CLK_SLEEP_DISABLE

__HAL_RCC_USART1_CLK_SLEEP_DISABLE

__HAL_RCC_TIM14_CLK_SLEEP_DISABLE

__HAL_RCC_TIM15_CLK_SLEEP_DISABLE

__HAL_RCC_TIM16_CLK_SLEEP_DISABLE

__HAL_RCC_TIM17_CLK_SLEEP_DISABLE

__HAL_RCC_ADC_CLK_SLEEP_DISABLE

APB1_GRP2 Peripheral Force Release Reset

__HAL_RCC_APB1_GRP2_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_TIM1_FORCE_RESET

__HAL_RCC_SPI1_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_TIM14_FORCE_RESET

__HAL_RCC_TIM15_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_ADC_FORCE_RESET

__HAL_RCC_APB1_GRP2_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_TIM1_RELEASE_RESET

__HAL_RCC_SPI1_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_TIM14_RELEASE_RESET

__HAL_RCC_TIM15_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

__HAL_RCC_ADC_RELEASE_RESET

RCC Clock Configuration

__HAL_RCC_HSI_ENABLE

Description:

- Macros to enable the Internal High Speed 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). 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.

__HAL_RCC_HSI_DISABLE

Description:

- Macros to disable the Internal High Speed oscillator (HSI).

Return value:

- None

Notes:

- 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. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal High Speed 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 127.

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 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_HSI_CONFIG

Description:

- Macro to configure the HSISYS clock.

Parameters:

- __HSIDIV__: specifies the HSI16 division factor. This parameter can be one of the following values:
  - RCC_HSI_DIV1 HSI clock source is divided by 1
  - RCC_HSI_DIV2 HSI clock source is divided by 2
  - RCC_HSI_DIV4 HSI clock source is divided by 4
  - RCC_HSI_DIV8 HSI clock source is divided by 8
  - RCC_HSI_DIV16 HSI clock source is divided by 16
  - RCC_HSI_DIV32 HSI clock source is divided by 32
  - RCC_HSI_DIV64 HSI clock source is divided by 64
  - RCC_HSI_DIV128 HSI clock source is divided by 128

__HAL_RCC_GET_HSI_DIVIDER

Description:

- Macro to get the HSI divider.

Return value:

- The: HSI divider. The returned value can be one of the following:
  - RCC_CR_HSIDIV_1 HSI oscillator divided by 1
  - RCC_CR_HSIDIV_2 HSI oscillator divided by 2
  - RCC_CR_HSIDIV_4 HSI oscillator divided by 4 (default after reset)
  - RCC_CR_HSIDIV_8 HSI oscillator divided by 8
  - RCC_CR_HSIDIV_16 HSI oscillator divided by 16
  - RCC_CR_HSIDIV_32 HSI oscillator divided by 32
  - RCC_CR_HSIDIV_64 HSI oscillator divided by 64
  - RCC_CR_HSIDIV_128 HSI oscillator divided by 128

__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.

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 modes. This function reset the CSSON bit, so if the clock security system(CSS) was previously enabled you have to enable it again after calling this function.

__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 Turn ON the LSE oscillator.
  - RCC_LSE_BYPASS LSE oscillator bypassed with external clock.

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. 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_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_HSI HSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSE HSE oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_HSIUSB48 HSIUSB48 oscillator is used as system clock source.(*)
  - RCC_SYSCLKSOURCE_LSI LSI oscillator is used as system clock source.
  - RCC_SYSCLKSOURCE_LSE LSE oscillator is used as system clock source.

Return value:

- None

Notes:

- (*) peripheral not available on all devices

__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_HSI HSI used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSE HSE used as system clock.
  - RCC_SYSCLKSOURCE_STATUS_HSIUSB48 HSIUSB48 used as system clock.(*)
  - RCC_SYSCLKSOURCE_STATUS_LSI LSI used as system clock source.
  - RCC_SYSCLKSOURCE_STATUS_LSE LSE used as system clock source.

Notes:

- (*) peripheral not available on all devices

__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

__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_HSI HSI clock selected as MCO source
  - RCC_MCO1SOURCE_HSE HSE 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
  - RCC_MCODIV_32 MCO clock source is divided by 32
  - RCC_MCODIV_64 MCO clock source is divided by 64
  - RCC_MCODIV_128 MCO clock source is divided by 128
  - RCC_MCODIV_256 MCO clock source is divided by 256 (*)
  - RCC_MCODIV_512 MCO clock source is divided by 512 (*)
  - RCC_MCODIV_1024 MCO clock source is divided by 1024 (*)

Return value:

- None

Notes:

- (*) not available on all devices

__HAL_RCC_MCO2_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_MCO2SOURCE_NOCLOCK MCO output disabled
  - RCC_MCO2SOURCE_SYSCLK System clock selected as MCO source
  - RCC_MCO2SOURCE_HSI HSI clock selected as MCO source
  - RCC_MCO2SOURCE_HSE HSE clock selected as MCO source
  - RCC_MCO2SOURCE_LSI LSI clock selected as MCO source
  - RCC_MCO2SOURCE_LSE LSE clock selected as MCO source
  - RCC_MCO2SOURCE_HSI48 HSI48 clock selected as MCO2 source for devices with HSI48 (*)
- __MCODIV__: specifies the MCO clock prescaler. This parameter can be one of the following values:
  - RCC_MCO2DIV_1 MCO clock source is divided by 1
  - RCC_MCO2DIV_2 MCO clock source is divided by 2
  - RCC_MCO2DIV_4 MCO clock source is divided by 4
  - RCC_MCO2DIV_8 MCO clock source is divided by 8
  - RCC_MCO2DIV_16 MCO clock source is divided by 16
  - RCC_MCO2DIV_32 MCO clock source is divided by 32
  - RCC_MCO2DIV_64 MCO clock source is divided by 64
  - RCC_MCO2DIV_128 MCO clock source is divided by 128
  - RCC_MCO2DIV_256 MCO clock source is divided by 256 (*)
  - RCC_MCO2DIV_512 MCO clock source is divided by 512 (*)
  - RCC_MCO2DIV_1024 MCO clock source is divided by 1024 (*)

Return value:

- None

Notes:

- (*) not available on all devices

Flags

RCC_FLAG_HSIRDY

HSI Ready flag

RCC_FLAG_HSERDY

HSE Ready flag

RCC_FLAG_LSERDY

LSE Ready flag

RCC_FLAG_LSECSSD

LSE Clock Security System Interrupt flag

RCC_FLAG_LSIRDY

LSI Ready flag

RCC_FLAG_OBLRST

Option Byte Loader reset flag

RCC_FLAG_PINRST

PIN reset flag

RCC_FLAG_PWRRST

BOR or POR/PDR 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.

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_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48 (*)

Return value:

- None

Notes:

- (*) peripheral not available on all devices

__HAL_RCC_DISABLE_IT

Description:

- Disable RCC interrupt.

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_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48 (*)

Return value:

- None

Notes:

- (*) peripheral not available on all devices

__HAL_RCC_CLEAR_IT

Description:

- Clear RCC 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_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48 (*)

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_HSIRDY HSI ready interrupt
  - RCC_IT_HSERDY HSE ready interrupt
  - RCC_IT_CSS HSE Clock security system interrupt
  - RCC_IT_LSECSS LSE Clock security system interrupt
  - RCC_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48 (*)

Return value:

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

Notes:

- (*) peripheral not available on all devices

__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_HSIRDY HSI oscillator clock ready
  - RCC_FLAG_HSERDY HSE oscillator 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_IT_HSI48RDY HSI48 ready interrupt for devices with HSI48 (*)
  - RCC_FLAG_PWRRST BOR or POR/PDR reset
  - RCC_FLAG_OBLRST OBLRST reset
  - RCC_FLAG_PINRST Pin reset
  - 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).

Notes:

- (*) peripheral not available on all devices

RCC HCLK Clock Divider

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

RCC_HCLK_DIV64

HCLK divided by 64

RCC_HCLK_DIV128

HCLK divided by 128

RCC_HCLK_DIV256

HCLK divided by 256

RCC_HCLK_DIV512

HCLK divided by 512

RCC HCLK Clock Source

LL_RCC_HCLK_DIV_1

HCLK not divided

LL_RCC_HCLK_DIV_2

HCLK divided by 2

LL_RCC_HCLK_DIV_4

HCLK divided by 4

LL_RCC_HCLK_DIV_8

HCLK divided by 8

LL_RCC_HCLK_DIV_16

HCLK divided by 16

LL_RCC_HCLK_DIV_64

HCLK divided by 64

LL_RCC_HCLK_DIV_128

HCLK divided by 128

LL_RCC_HCLK_DIV_256

HCLK divided by 256

LL_RCC_HCLK_DIV_512

HCLK divided by 512

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

RCC_HSE_BYPASS

External clock source for HSE clock

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSICALIBRATION_DEFAULT

Default HSI calibration trimming value

HSI Div

RCC_HSI_DIV1

HSI clock is not divided

RCC_HSI_DIV2

HSI clock is divided by 2

RCC_HSI_DIV4

HSI clock is divided by 4

RCC_HSI_DIV8

HSI clock is divided by 8

RCC_HSI_DIV16

HSI clock is divided by 16

RCC_HSI_DIV32

HSI clock is divided by 32

RCC_HSI_DIV64

HSI clock is divided by 64

RCC_HSI_DIV128

HSI clock is divided by 128

Interrupts

RCC_IT_LSIRDY

LSI Ready Interrupt flag

RCC_IT_LSERDY

LSE Ready Interrupt flag

RCC_IT_HSIRDY

HSI Ready Interrupt flag

RCC_IT_HSERDY

HSE Ready Interrupt flag

RCC_IT_CSS

HSE Clock Security System Interrupt flag

RCC_IT_LSECSS

LSE Clock Security System Interrupt flag

IOPORT 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_GPIOF_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_GPIOF_IS_CLK_DISABLED

IOPORT 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_GPIOF_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_GPIOC_CLK_DISABLE

__HAL_RCC_GPIOD_CLK_DISABLE

__HAL_RCC_GPIOF_CLK_DISABLE

IOPORT Clock Sleep Enabled or Disabled Status

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_ENABLED

__HAL_RCC_GPIOA_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOB_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOC_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOD_IS_CLK_SLEEP_DISABLED

__HAL_RCC_GPIOF_IS_CLK_SLEEP_DISABLED

IOPORT 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_GPIOF_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_GPIOF_CLK_SLEEP_DISABLE

IOPORT Force Release Reset

__HAL_RCC_IOP_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_GPIOC_FORCE_RESET

__HAL_RCC_GPIOD_FORCE_RESET

__HAL_RCC_GPIOF_FORCE_RESET

__HAL_RCC_IOP_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_GPIOC_RELEASE_RESET

__HAL_RCC_GPIOD_RELEASE_RESET

__HAL_RCC_GPIOF_RELEASE_RESET

LSE Drive Configuration

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

LSE clock activation

RCC_LSE_BYPASS

External clock source for LSE clock

LSI Config

RCC_LSI_OFF

LSI clock deactivation

RCC_LSI_ON

LSI clock activation

MCO1 Clock Prescaler

RCC_MCODIV_1

MCO not divided

RCC_MCODIV_2

MCO divided by 2

RCC_MCODIV_4

MCO divided by 4

RCC_MCODIV_8

MCO divided by 8

RCC_MCODIV_16

MCO divided by 16

RCC_MCODIV_32

MCO divided by 32

RCC_MCODIV_64

MCO divided by 64

RCC_MCODIV_128

MCO divided by 128

RCC_MCODIV_256

MCO divided by 256

RCC_MCODIV_512

MCO divided by 512

RCC_MCODIV_1024

MCO divided by 1024

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

MCO1 output disabled, no clock on MCO1

RCC_MCO1SOURCE_SYSCLK

SYSCLK selection as MCO1 source

RCC_MCO1SOURCE_HSI

HSI selection as MCO1 source

RCC_MCO1SOURCE_HSE

HSE selection as MCO1 source

RCC_MCO1SOURCE_LSI

LSI selection as MCO1 source

RCC_MCO1SOURCE_LSE

LSE selection as MCO1 source

MCO2 Clock Prescaler

RCC_MCO2DIV_1

MCO not divided

RCC_MCO2DIV_2

MCO divided by 2

RCC_MCO2DIV_4

MCO divided by 4

RCC_MCO2DIV_8

MCO divided by 8

RCC_MCO2DIV_16

MCO divided by 16

RCC_MCO2DIV_32

MCO divided by 32

RCC_MCO2DIV_64

MCO divided by 64

RCC_MCO2DIV_128

MCO divided by 128

RCC_MCO2DIV_256

MCO divided by 256

RCC_MCO2DIV_512

MCO divided by 512

RCC_MCO2DIV_1024

MCO divided by 1024

MCO2 Clock Source

RCC_MCO2SOURCE_NOCLOCK

MCO2 output disabled, no clock on MCO2

RCC_MCO2SOURCE_SYSCLK

SYSCLK selection as MCO2 source

RCC_MCO2SOURCE_HSI

HSI selection as MCO2 source

RCC_MCO2SOURCE_HSE

HSE selection as MCO2 source

RCC_MCO2SOURCE_LSI

LSI selection as MCO2 source

RCC_MCO2SOURCE_LSE

LSE selection as MCO2 source

MCO Index

RCC_MCO1_PA8

RCC_MCO1_PA9

RCC_MCO1_PF2

RCC_MCO1

Alias for compatibility

RCC_MCO2_PA8

RCC_MCO2_PA10

RCC_MCO2_PA14

RCC_MCO2_PA15

RCC_MCO2_PB2

RCC_MCO2

Alias for compatibility

RCC_MCO

MCO1 to be compliant with other families with 1 MCO

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 RTC Clock Configuration

__HAL_RCC_RTC_ENABLE

Description:

- Macros to enable or disable the RTC clock.

Return value:

- None

Notes:

- These macros must be used after the RTC clock source was selected.

__HAL_RCC_RTC_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_NONE 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 RTC domain and write access is denied to this domain after reset, you have to enable write access using the Power RTC 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 RTC 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_NONE 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

RTC Clock Source

RCC_RTCCLKSOURCE_NONE

No clock configured for RTC

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

RCC RTC Domain Reset

__HAL_RCC_BACKUPRESET_FORCE

Description:

- Macros to force or release the RTC domain reset.

Return value:

- None

Notes:

- This function resets the RTC peripheral and the RTC clock source selection in RCC_CSR1 register.

__HAL_RCC_BACKUPRESET_RELEASE

System Clock Source

RCC_SYSCLKSOURCE_HSI

HSI selection as system clock

RCC_SYSCLKSOURCE_HSE

HSE selection as system clock

RCC_SYSCLKSOURCE_LSI

LSI selection as system clock

RCC_SYSCLKSOURCE_LSE

LSE selection as system clock

System Clock Source Status

RCC_SYSCLKSOURCE_STATUS_HSI

HSI used as system clock

RCC_SYSCLKSOURCE_STATUS_HSE

HSE used as system clock

RCC_SYSCLKSOURCE_STATUS_LSI

LSI used as system clock

RCC_SYSCLKSOURCE_STATUS_LSE

LSE 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 SYS Clock Divider

RCC_SYSCLK_DIV1

SYSCLK not divided

RCC_SYSCLK_DIV2

SYSCLK is divided by 2

RCC_SYSCLK_DIV3

SYSCLK is divided by 3

RCC_SYSCLK_DIV4

SYSCLK is divided by 4

RCC_SYSCLK_DIV5

SYSCLK is divided by 5

RCC_SYSCLK_DIV6

SYSCLK is divided by 6

RCC_SYSCLK_DIV7

SYSCLK is divided by 7

RCC_SYSCLK_DIV8

SYSCLK is divided by 8

Timeout Values

RCC_LSE_TIMEOUT_VALUE

29 HAL RCC Extension Driver

29.1 RCCEx Firmware driver registers structures

29.1.1 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32c0xx_hal_rcc_ex.h

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t HSIKerClockDivider***
- ***uint32_t Usart1ClockSelection***
- ***uint32_t I2c1ClockSelection***
- ***uint32_t I2s1ClockSelection***
- ***uint32_t Fdcan1ClockSelection***
- ***uint32_t AdcClockSelection***
- ***uint32_t RTCClockSelection***

Field Documentation

- ***uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection***
  The Extended Clock to be configured. This parameter can be a value of ***RCCEx_Periph_Clock_Selection***
- ***uint32_t RCC_PeriphCLKInitTypeDef::HSIKerClockDivider***
  Specifies HSI Kernel divider. This parameter can be a value of ***RCCEx_HSIKER_Div***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Usart1ClockSelection***
  Specifies USART1 clock source. This parameter can be a value of ***RCCEx_USART1_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::I2s1ClockSelection***
  Specifies I2S1 clock source This parameter can be a value of ***RCCEx_I2S1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::Fdcan1ClockSelection***
  Specifies FDCAN1 clock source. This parameter can be a value of ***RCCEx_FDCAN1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::AdcClockSelection***
  Specifies ADC interface clock source This parameter can be a value of ***RCCEx_ADC_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection***
  Specifies RTC clock source. This parameter can be a value of ***RCC_RTC_Clock_Source***

29.2 RCCEx Firmware driver API description

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

29.2.1 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 @ref HAL_RCCEx_PeriphCLKConfig() is used to select the RTC clock source; as consequence RTC registers and RCC_CSR1 register are set to their reset values.*

This section contains the following APIs:

- ***HAL_RCCEx_PeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKConfig()***
- ***HAL_RCCEx_GetPeriphCLKFreq()***

29.2.2 Extended clock management functions

This subsection provides a set of functions allowing to control the activation or deactivation of LSE CSS, Low speed clock output and clock after wake-up from STOP mode.

This section contains the following APIs:

- *HAL_RCCEx_EnableLSCO()*
- *HAL_RCCEx_DisableLSCO()*

29.2.3 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (const RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Initialize the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.

Parameters

- **PeriphClkInit:** pointer to a RCC_PeriphCLKInitTypeDef structure that contains a field PeriphClockSelection which can be a combination of the following values:
  - RCC_PERIPHCLK_RTC RTC peripheral clock
  - RCC_PERIPHCLK_ADC ADC peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2S1 I2S1 peripheral clock
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USB USB peripheral clock (*)
  - RCC_PERIPHCLK_FDCAN1 FDCAN1 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 RTC domain is enabled.
- (*) not available on all devices

HAL_RCCEx_GetPeriphCLKConfig

Function name

void HAL_RCCEx_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)

Function description

Get the RCC_ClkInitStruct according to the internal RCC configuration registers.

Parameters

- **PeriphClkInit:** pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks: I2C1, I2S1, USART1, RTC, ADC,

Return values

- **None:**

HAL_RCCEx_GetPeriphCLKFreq

Function name

uint32_t HAL_RCCEx_GetPeriphCLKFreq (uint32_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_RTC RTC peripheral clock
  - RCC_PERIPHCLK_ADC ADC peripheral clock
  - RCC_PERIPHCLK_I2C1 I2C1 peripheral clock
  - RCC_PERIPHCLK_I2S1 I2S1 peripheral clock
  - RCC_PERIPHCLK_USART1 USART1 peripheral clock
  - RCC_PERIPHCLK_USB USB peripheral clock (*)
  - RCC_PERIPHCLK_FDCAN1 FDCAN1 peripheral clock (*)

Return values

- **Frequency:** in Hz

Notes

- Return 0 if peripheral clock identifier not managed by this API
- (*) not available on all 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:**

29.3 RCCEx Firmware driver defines

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

29.3.1 RCCEx

RCCEx

RCC ADC Clock Source

RCC_ADCCLKSOURCE_SYSCLK

SYSCLK used as ADC clock

RCC_ADCCLKSOURCE_HSIKER

HSI kernel used as ADC clock

RCCEx Exported Macros

__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_HSIKER HSI Kernel selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

__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_HSIKER HSI Kernel selected as I2C1 clock
  - RCC_I2C1CLKSOURCE_SYSCLK System Clock selected as I2C1 clock

__HAL_RCC_I2S1_CONFIG

Description:

- Macro to configure the I2S1 clock (I2S1CLK).

Parameters:

- __I2S1_CLKSOURCE__: specifies the I2S1 clock source. This parameter can be one of the following values:
  - RCC_I2S1CLKSOURCE_SYSCLK System Clock selected as I2S1 clock
  - RCC_I2S1CLKSOURCE_HSIKER HSI Kernel Clock selected as I2S1 clock
  - RCC_I2S1CLKSOURCE_EXT External clock selected as I2S1 clock

__HAL_RCC_GET_I2S1_SOURCE

Description:

- Macro to get the I2S1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_I2S1CLKSOURCE_SYSCLK System Clock selected as I2S1 clock
  - RCC_I2S1CLKSOURCE_HSIKER HSI Kernel Clock selected as I2S1 clock
  - RCC_I2S1CLKSOURCE_EXT External clock selected as I2S1 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_PCLK1 PCLK1 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSIKER HSI Kernel selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__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_PCLK1 PCLK1 selected as USART1 clock
  - RCC_USART1CLKSOURCE_HSIKER HSI Kernel selected as USART1 clock
  - RCC_USART1CLKSOURCE_SYSCLK System Clock selected as USART1 clock
  - RCC_USART1CLKSOURCE_LSE LSE selected as USART1 clock

__HAL_RCC_FDCAN1_CONFIG

Description:

- Macro to configure the FDCAN1 clock (FDCAN1CLK).

Parameters:

- __FDCAN1_CLKSOURCE__: specifies the FDCAN1 clock source. This parameter can be one of the following values:
  - RCC_FDCAN1CLKSOURCE_PCLK1 APB Clock selected as FDCAN1 clock
  - RCC_FDCAN1CLKSOURCE_HSIKER HSI Kernel Clock selected as FDCAN1 clock
  - RCC_FDCAN1CLKSOURCE_HSE HSE clock selected as FDCAN1 clock

__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_PCLK1 APB Clock selected as FDCAN1 clock
  - RCC_FDCAN1CLKSOURCE_HSIKER HSI Kernel Clock selected as FDCAN1 clock
  - RCC_FDCAN1CLKSOURCE_HSE HSE clock selected as FDCAN1 clock

__HAL_RCC_ADC_CONFIG

Description:

- Macro to configure the ADC interface clock.

Parameters:

- __ADC_CLKSOURCE__: specifies the ADC digital interface clock source. This parameter can be one of the following values:
  - RCC_ADCCLKSOURCE_SYSCLK System Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_HSIKER HSI Kernel Clock selected as ADC clock

__HAL_RCC_GET_ADC_SOURCE

Description:

- Macro to get the ADC clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_ADCCLKSOURCE_SYSCLK System Clock selected as ADC clock
  - RCC_ADCCLKSOURCE_HSIKER HSI Kernel Clock selected as ADC clock

__HAL_RCC_HSIKER_CONFIG

Description:

- Macro to configure the HSIKER clock.

Parameters:

- __HSIKERDIV__: specifies the HSI Kernel division factor. This parameter can be one of the following values:
  - RCC_HSIKER_DIV1 HSI clock source is divided by 1
  - RCC_HSIKER_DIV2 HSI clock source is divided by 2
  - RCC_HSIKER_DIV3 HSI clock source is divided by 3
  - RCC_HSIKER_DIV4 HSI clock source is divided by 4
  - RCC_HSIKER_DIV5 HSI clock source is divided by 5
  - RCC_HSIKER_DIV6 HSI clock source is divided by 6
  - RCC_HSIKER_DIV7 HSI clock source is divided by 7
  - RCC_HSIKER_DIV8 HSI clock source is divided by 8

__HAL_RCC_GET_HSIKER_DIVIDER

Description:

- Macro to get the HSIKER divider.

Return value:

- The: HSI Kernel divider. The returned value can be one of the following:
  - RCC_HSIKER_DIV1 HSI oscillator divided by 1
  - RCC_HSIKER_DIV2 HSI oscillator divided by 2
  - RCC_HSIKER_DIV3 HSI oscillator divided by 3 (default after reset)
  - RCC_HSIKER_DIV4 HSI oscillator divided by 4
  - RCC_HSIKER_DIV5 HSI oscillator divided by 5
  - RCC_HSIKER_DIV6 HSI oscillator divided by 6
  - RCC_HSIKER_DIV7 HSI oscillator divided by 7
  - RCC_HSIKER_DIV8 HSI oscillator divided by 8

FDCAN1 Clock Source

RCC_FDCAN1CLKSOURCE_PCLK1

RCC_FDCAN1CLKSOURCE_HSIKER

RCC_FDCAN1CLKSOURCE_HSE

HSIKER Div

RCC_HSIKER_DIV1

HSI Kernek clock is not divided

RCC_HSIKER_DIV2

HSI Kernel clock is divided by 2

RCC_HSIKER_DIV3

HSI Kernel clock is divided by 3

RCC_HSIKER_DIV4

HSI Kernel clock is divided by 4

RCC_HSIKER_DIV5

HSI Kernel clock is divided by 5

RCC_HSIKER_DIV6

HSI Kernel clock is divided by 6

RCC_HSIKER_DIV7

HSI Kernel clock is divided by 7

RCC_HSIKER_DIV8

HSI Kernel clock is divided by 8

RCC I2C1 Clock Source

RCC_I2C1CLKSOURCE_PCLK1

APB clock selected as I2C1 clock

RCC_I2C1CLKSOURCE_SYSCLK

SYSCLK clock selected as I2C1 clock

RCC_I2C1CLKSOURCE_HSIKER

HSI Kernel clock selected as I2C1 clock

RCC I2S1 Clock Source

RCC_I2S1CLKSOURCE_SYSCLK

SYSCLK clock selected as I2S1 clock

RCC_I2S1CLKSOURCE_HSIKER

HSI Kernel clock selected as I2S1 clock

RCC_I2S1CLKSOURCE_EXT

External I2S clock source selected as I2S1 clock

Low Speed Clock Source

RCC_LSCOSOURCE_LSI

LSI selection for low speed clock output

RCC_LSCOSOURCE_LSE

LSE selection for low speed clock output

Periph Clock Selection

RCC_PERIPHCLK_USART1

RCC_PERIPHCLK_I2C1

RCC_PERIPHCLK_I2S1

RCC_PERIPHCLK_FDCAN1

RCC_PERIPHCLK_ADC

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_HSIKER

RCC USART1 Clock Source

RCC_USART1CLKSOURCE_PCLK1

APB clock selected as USART 1 clock

RCC_USART1CLKSOURCE_SYSCLK

SYSCLK clock selected as USART 1 clock

RCC_USART1CLKSOURCE_HSIKER

HSI Kernel clock selected as USART 1 clock

RCC_USART1CLKSOURCE_LSE

LSE clock selected as USART 1 clock

30 HAL RTC Generic Driver

30.1 RTC Firmware driver registers structures

30.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32c0xx_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***

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***

30.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32c0xx_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 parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity
- ***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***
  Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of ***RTC_DayLightSaving_Definitions***
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  Specifies RTC_StoreOperation value to be written in the BKP bit in CR register to store the operation. This parameter can be a value of ***RTC_StoreOperation_Definitions***

30.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32c0xx_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

30.1.4 RTC_AlarmTypeDef

RTC_AlarmTypeDef is defined in the stm32c0xx_hal_rtc.h

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***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. This parameter can be a value of ***RTC_Alarm_Sub_Seconds_Masks_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::Alarm***
  Specifies the alarm . This parameter can be a value of ***RTC_Alarms_Definitions***

30.1.5 __RTC_HandleTypeDef

__RTC_HandleTypeDef is defined in the stm32c0xx_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(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***RTC_TypeDef* __RTC_HandleTypeDef::Instance***
  Register 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::MspInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp Init callback
- ***void(* __RTC_HandleTypeDef::MspDeInitCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC Msp DeInit callback

30.2 RTC Firmware driver API description

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

30.2.1 How to use RTC Driver

- 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.

30.2.2 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 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 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

The compilation define USE_RTC_REGISTER_CALLBACKS when set to 1 allows the user to configure dynamically the driver callbacks. Use Function HAL_RTC_RegisterCallback() to register an interrupt callback.

Function HAL_RTC_RegisterCallback() allows to register following callbacks:

- AlarmAEventCallback : RTC Alarm A Event callback.
- : RTC TimeStamp Event callback.
- MspInitCallback : RTC MspInit.
- MspDeInitCaTimeStampEventCallbackIlback : RTC MspDeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function. Use function HAL_RTC_UnRegisterCallback() to reset a callback to the default weak function. HAL_RTC_UnRegisterCallback() takes as parameters the HAL peripheral handle, and the Callback ID. This function allows to reset following callbacks:
- AlarmAEventCallback : RTC Alarm A Event callback.
- TimeStampEventCallback : RTC TimeStamp Event callback.
- MspInitCallback : RTC MspInit callback.
- MspDeInitCallback : RTC MspDeInit callback.

By default, after the HAL_RTC_Init() and when the state is HAL_RTC_STATE_RESET, all callbacks are set to the corresponding weak functions : examples AlarmAEventCallback(), TimeStampEventCallback(). Exception done for MspInit and MspDeInit callbacks that are reset to the legacy weak function in the HAL_RTC_Init()/ HAL_RTC_DeInit() only when these callbacks are null (not registered beforehand). If not, MspInit or MspDeInit are not null, HAL_RTC_Init()/HAL_RTC_DeInit() keep and use the user MspInit/MspDeInit callbacks (registered beforehand)

Callbacks can be registered/unregistered in HAL_RTC_STATE_READY state only. Exception done MspInit/ MspDeInit that can be registered/unregistered in HAL_RTC_STATE_READY or HAL_RTC_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_RTC_RegisterCallback() before calling HAL_RTC_DeInit() or HAL_RTC_Init() function.

When The compilation define USE_HAL_RTC_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.

30.2.3 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()***

30.2.4 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_GetTime()***
- ***HAL_RTC_SetDate()***
- ***HAL_RTC_GetDate()***

30.2.5 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()***

30.2.6 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()***

30.2.7 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

30.2.8 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_TIMESTAMP_EVENT_CB_ID TimeStamp Event 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:
  - HAL_RTC_ALARM_A_EVENT_CB_ID Alarm A Event Callback ID
  - HAL_RTC_TIMESTAMP_EVENT_CB_ID TimeStamp Event 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
- **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_RTC_GetTime

Function name

HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)

Function description

Get RTC current time.

Parameters

- **hrtc:** RTC handle
- **sTime:** 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:** 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

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:** 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_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:** 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

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_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
- **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_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
- **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

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.
- The application must ensure that the EXTI RTC interrupt line is enabled.

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

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
- **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_RTC_AlarmIRQHandler

Function name

void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handle Alarm interrupt request.

Parameters

- **hrtc:** RTC handle

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (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 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

30.3 RTC Firmware driver defines

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

30.3.1 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 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 Alarm comparison. 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_TSOVF

Clear Time-stamp overflow flag

RTC_CLEAR_TSF

Clear Time-stamp 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_IS_CALENDAR_INITIALIZED

Description:

- Check whether the RTC Calendar is initialized.

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

__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

__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_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 Interrupts Definitions

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

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

Return value:

- None

__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

Return value:

- None

__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_ALRAWF

Return value:

- None

__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

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

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_ALRAWF

Alarm A write flag

RTC_FLAG_TSOVF

Time-stamp overflow flag

RTC_FLAG_TSF

Time-stamp flag

RTC_FLAG_ALRAF

Alarm A flag

RTC Flag Mask (5bits) describe __HAL_RTC_GET_FLAG()

RTC_FLAG_MASK

RTC flags mask (5bits)

RTC Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

RTC Input Parameter Format Definitions

RTC_FORMAT_BIN

RTC_FORMAT_BCD

RTC Interrupts Definitions

RTC_IT_TS

Enable Timestamp Interrupt

RTC_IT_ALRA

Enable Alarm A 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

31 HAL RTC Extension Driver

31.1 RTCEx Firmware driver API description

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

31.1.1 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.

Outputs configuration

The RTC has 2 different outputs:

- RTC_ALARM: this output is used to manage the RTC Alarm A. 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.

31.1.2 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_GetTimeStamp()***
- ***HAL_RTCEx_TimeStampEventCallback()***
- ***HAL_RTCEx_TimeStampIRQHandler()***
- ***HAL_RTCEx_PollForTimeStampEvent()***

31.1.3 Extended Peripheral Control functions

This subsection provides functions allowing to

- Set the Coarse calibration parameters.
- Deactivate the Coarse calibration parameters
- Set the Smooth calibration parameters.
- 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_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*
- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*

31.1.4 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.
- The application must ensure that the EXTI RTC interrupt line is enabled.

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_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_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 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_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.

31.2 RTCEx Firmware driver defines

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

31.2.1 RTCEx

RTCEx

RTCEx Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

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:

- None

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_TSOVF Clear Time-stamp overflow flag
  - RTC_CLEAR_TSF Clear Time-stamp 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:
  - 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_ALRAWF Alarm A write flag
  - RTC_FLAG_TSOVF Time-stamp overflow flag
  - RTC_FLAG_TSF Time-stamp flag
  - RTC_FLAG_ALRAF Alarm A flag

Return value:

- None

Private macros to check input parameters

IS_TIMESTAMP_EDGE

IS_RTC_TIMESTAMP_PIN

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SHIFT_ADD1S

IS_RTC_CALIB_OUTPUT

RTCEx Output Selection Definition

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTCEx Smooth calib Minus pulses Definitions

IS_RTC_SMOOTH_CALIB_MINUS

RTCEx Smooth calib period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, Smooth calibation period is 32s, else 2exp20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, Smooth calibation period is 16s, else 2exp19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, Smooth calibation 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]

RTCEx SubtractFraction Of Second Value

IS_RTC_SHIFT_SUBFS

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:

- None

__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:

- None

__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:

- None

__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_TIMESTAMP_EXTI_ENABLE_IT

Description:

- Enable interrupt on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_DISABLE_IT

Description:

- Disable interrupt on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_ENABLE_EVENT

Description:

- Enable event on the RTC Timestamp associated Exti line.

Return value:

- None

__HAL_RTC_TIMESTAMP_EXTI_DISABLE_EVENT

Description:

- Disable event on the RTC Timestamp associated Exti line.

Return value:

- None

RTCEx TimeStamp Pin Selection

RTC_TIMESTAMPPIN_DEFAULT

RTCEx Time Stamp Edges definition

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

32 HAL SMARTCARD Generic Driver

32.1 SMARTCARD Firmware driver registers structures

32.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32c0xx_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***.

32.1.2 SMARTCARD_AdvFeatureInitTypeDef

SMARTCARD_AdvFeatureInitTypeDef is defined in the stm32c0xx_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***.

32.1.3 __SMARTCARD_HandleTypeDef

__SMARTCARD_HandleTypeDef is defined in the stm32c0xx_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

32.2 SMARTCARD Firmware driver API description

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

32.2.1 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

32.2.2 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.

32.2.3 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()***

32.2.4 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()***

32.2.5 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()***

32.2.6 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

32.3 SMARTCARD Firmware driver defines

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

32.3.1 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

33 HAL SMARTCARD Extension Driver

33.1 SMARTCARDEx Firmware driver API description

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

33.1.1 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.

33.1.2 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()***

33.1.3 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()***

33.1.4 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()***

33.1.5 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 0x0FFFFFFF.

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

33.2 SMARTCARDEx Firmware driver defines

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

33.2.1 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

34 HAL SMBUS Generic Driver

34.1 SMBUS Firmware driver registers structures

34.1.1 SMBUS_InitTypeDef

SMBUS_InitTypeDef is defined in the stm32c0xx_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

34.1.2 __SMBUS_HandleTypeDef

__SMBUS_HandleTypeDef is defined in the stm32c0xx_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

34.2 SMBUS Firmware driver API description

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

34.2.1 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

34.2.2 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()*

34.2.3 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()*

34.2.4 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()*

34.2.5 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

34.3 SMBUS Firmware driver defines

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

34.3.1 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_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

35 HAL SMBUS Extension Driver

35.1 SMBUSEx Firmware driver API description

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

35.1.1 SMBUS peripheral Extended features

Comparing to other previous devices, the SMBUS interface for STM32C0xx devices contains the following additional features

- Disable or enable wakeup from Stop mode(s)
- Disable or enable Fast Mode Plus

35.1.2 How to use this driver

35.1.3 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()***

35.1.4 Fast Mode Plus Functions

This section provides functions allowing to:

- Configure Fast Mode Plus

This section contains the following APIs:

- ***HAL_SMBUSEx_EnableFastModePlus()***
- ***HAL_SMBUSEx_DisableFastModePlus()***

35.1.5 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_EnableFastModePlus

Function name

void HAL_SMBUSEx_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the SMBUS fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the SMBUS Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be enabled on all selected I2C1 pins using SMBUS_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be enabled only by using SMBUS_FASTMODEPLUS_I2C2 parameter.

HAL_SMBUSEx_DisableFastModePlus

Function name

void HAL_SMBUSEx_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the SMBUS fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** Selects the pin. This parameter can be one of the SMBUS Extended Fast Mode Plus values

Return values

- **None:**

Notes

- For I2C1, fast mode plus driving capability can be disabled on all selected I2C1 pins using SMBUS_FASTMODEPLUS_I2C1 parameter or independently on each one of the following pins PB6, PB7, PB8 and PB9.
- For remaining I2C1 pins (PA14, PA15...) fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C1 parameter.
- For all I2C2 pins fast mode plus driving capability can be disabled only by using SMBUS_FASTMODEPLUS_I2C2 parameter.

35.2 SMBUSEx Firmware driver defines

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

35.2.1 SMBUSEx

SMBUSEx

SMBUS Extended Fast Mode Plus

SMBUS_FMP_NOT_SUPPORTED

Fast Mode Plus not supported

SMBUS_FASTMODEPLUS_PA9

Enable Fast Mode Plus on PA9

SMBUS_FASTMODEPLUS_PA10

Enable Fast Mode Plus on PA10

SMBUS_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

SMBUS_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

SMBUS_FASTMODEPLUS_PB8

Enable Fast Mode Plus on PB8

SMBUS_FASTMODEPLUS_PB9

Enable Fast Mode Plus on PB9

SMBUS_FASTMODEPLUS_I2C1

Enable Fast Mode Plus on I2C1 pins

SMBUS_FASTMODEPLUS_I2C2

Enable Fast Mode Plus on I2C2 pins

36 HAL SPI Generic Driver

36.1 SPI Firmware driver registers structures

36.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32c0xx_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***

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 = 1 and Max_Data = 65535
- ***uint32_t SPI_InitTypeDef::CRCLength***
  Specifies the CRC Length used for the CRC calculation. CRC Length is only used with Data8 and Data16, not other data size 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 NSSP bit in the SPIx_CR2 register and it takes effect only if the SPI interface is configured as Motorola SPI master (FRF=0) with capture on the first edge (SPIx_CR1 CPHA = 0, CPOL setting is ignored)..

36.1.2 __SPI_HandleTypeDef

__SPI_HandleTypeDef is defined in the stm32c0xx_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(* 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::MspInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp Init callback
- ***void(* __SPI_HandleTypeDef::MspDeInitCallback)(struct __SPI_HandleTypeDef *hspi)***
  SPI Msp DeInit callback

36.2 SPI Firmware driver API description

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

36.2.1 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
      - 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(or _rx) 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.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
   a. Master 2Lines RxOnly
   b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

Master Receive mode restriction:

1. In Master unidirectional receive-only mode (MSTR =1, BIDIMODE=0, RXONLY=1) or bidirectional receive mode (MSTR=1, BIDIMODE=1, BIDIOE=0), to ensure that the SPI does not initiate a new transfer the following procedure has to be respected:
   a. HAL_SPI_DeInit()
   b. HAL_SPI_Init()

Data buffer address alignment restriction:

1. In case more than 1 byte is requested to be transferred, the HAL SPI uses 16-bit access for data buffer. But there is no support for unaligned accesses on the Cortex-M0 processor. So, if the user wants to transfer more than 1 byte, it shall ensure that 16-bit aligned address is used for:
   a. pData parameter in HAL_SPI_Transmit(), HAL_SPI_Transmit_IT(), HAL_SPI_Receive() and HAL_SPI_Receive_IT()
   b. pTxData and pRxData parameters in HAL_SPI_TransmitReceive() and HAL_SPI_TransmitReceive_IT()
2. There is no such restriction when going through DMA by using HAL_SPI_Transmit_DMA(), HAL_SPI_Receive_DMA() and HAL_SPI_TransmitReceive_DMA().

Callback registration:

1. The compilation flag USE_HAL_SPI_REGISTER_CALLBACKS when set to 1U 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
   - 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
   - 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 (surcharged) callbacks are used.

Using the HAL it is not possible to reach all supported SPI frequency with the different SPI Modes, the following table resume the max SPI frequency reached with data size 8bits/16bits, according to frequency of the APBx Peripheral Clock (fPCLK) used by the SPI instance.

36.2.2 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
- 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()***

36.2.3 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:
   - 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 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()*

36.2.4 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()*

36.2.5 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

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

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

- In case of MASTER mode and SPI_DIRECTION_2LINES direction, hdmatx shall be defined.
- 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:** status

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:** status

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:** status

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 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_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_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

36.3 SPI Firmware driver defines

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

36.3.1 SPI

SPI

SPI BaudRate Prescaler

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 Length

SPI_CRC_LENGTH_DATASIZE

SPI_CRC_LENGTH_8BIT

SPI_CRC_LENGTH_16BIT

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 Direction Mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

SPI Error Code

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 RXNE/TXE/BSY/FTLVL/FRLVL Flag

HAL_SPI_ERROR_ABORT

Error during SPI Abort procedure

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, or 3 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, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to enable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT

Description:

- Disable the specified SPI interrupts.

Parameters:

- __HANDLE__: specifies the SPI handle. This parameter can be SPIx where x: 1, 2, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the interrupt source to disable. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

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, or 3 to select the SPI peripheral.
- __INTERRUPT__: specifies the SPI interrupt source to check. This parameter can be one of the following values:
  - SPI_IT_TXE: Tx buffer empty interrupt enable
  - SPI_IT_RXNE: RX buffer not empty interrupt enable
  - SPI_IT_ERR: Error interrupt enable

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, or 3 to select the SPI peripheral.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  - SPI_FLAG_RXNE: Receive buffer not empty flag
  - SPI_FLAG_TXE: Transmit buffer empty flag
  - SPI_FLAG_CRCERR: CRC error flag
  - SPI_FLAG_MODF: Mode fault flag
  - SPI_FLAG_OVR: Overrun flag
  - SPI_FLAG_BSY: Busy flag
  - SPI_FLAG_FRE: Frame format error flag
  - SPI_FLAG_FTLVL: SPI fifo transmission level
  - SPI_FLAG_FRLVL: SPI fifo reception level

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. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_MODFFLAG

Description:

- Clear the SPI MODF pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_OVRFLAG

Description:

- Clear the SPI OVR pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_CLEAR_FREFLAG

Description:

- Clear the SPI FRE pending flag.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE

Description:

- Enable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_DISABLE

Description:

- Disable the SPI peripheral.

Parameters:

- __HANDLE__: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI FIFO Reception Threshold

SPI_RXFIFO_THRESHOLD

SPI_RXFIFO_THRESHOLD_QF

SPI_RXFIFO_THRESHOLD_HF

SPI Flags Definition

SPI_FLAG_RXNE

SPI_FLAG_TXE

SPI_FLAG_BSY

SPI_FLAG_CRCERR

SPI_FLAG_MODF

SPI_FLAG_OVR

SPI_FLAG_FRE

SPI_FLAG_FTLVL

SPI_FLAG_FRLVL

SPI_FLAG_MASK

SPI Interrupt Definition

SPI_IT_TXE

SPI_IT_RXNE

SPI_IT_ERR

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_ENABLE

SPI_NSS_PULSE_DISABLE

SPI Reception FIFO Status Level

SPI_FRLVL_EMPTY

SPI_FRLVL_QUARTER_FULL

SPI_FRLVL_HALF_FULL

SPI_FRLVL_FULL

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 Transmission FIFO Status Level

SPI_FTLVL_EMPTY

SPI_FTLVL_QUARTER_FULL

SPI_FTLVL_HALF_FULL

SPI_FTLVL_FULL

37 HAL SPI Extension Driver

37.1 SPIEx Firmware driver API description

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

37.1.1 IO operation functions

This subsection provides a set of extended functions to manage the SPI data transfers.

1. Rx data flush function:
   – HAL_SPIEx_FlushRxFifo()

This section contains the following APIs:

- ***HAL_SPIEx_FlushRxFifo()***

37.1.2 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

38 HAL TIM Generic Driver

38.1 TIM Firmware driver registers structures

38.1.1 TIM_Base_InitTypeDef

TIM_Base_InitTypeDef is defined in the stm32c0xx_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
- ***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.
- ***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***

38.1.2 TIM_OC_InitTypeDef

TIM_OC_InitTypeDef is defined in the stm32c0xx_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

- ***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.

38.1.3 TIM_OnePulse_InitTypeDef

TIM_OnePulse_InitTypeDef is defined in the stm32c0xx_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
- ***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

38.1.4 TIM_IC_InitTypeDef

TIM_IC_InitTypeDef is defined in the stm32c0xx_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

38.1.5 TIM_Encoder_InitTypeDef

TIM_Encoder_InitTypeDef is defined in the stm32c0xx_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

38.1.6 TIM_ClockConfigTypeDef

TIM_ClockConfigTypeDef is defined in the stm32c0xx_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

38.1.7 TIM_ClearInputConfigTypeDef

TIM_ClearInputConfigTypeDef is defined in the stm32c0xx_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

38.1.8 TIM_MasterConfigTypeDef

TIM_MasterConfigTypeDef is defined in the stm32c0xx_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.

38.1.9 TIM_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32c0xx_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

38.1.10 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32c0xx_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***

38.1.11 __TIM_HandleTypeDef

__TIM_HandleTypeDef is defined in the stm32c0xx_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***

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

38.2 TIM Firmware driver API description

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

38.2.1 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

38.2.2 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 and HAL_TIM_OC_ConfigChannel: 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.

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.

38.2.3 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()***

38.2.4 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()***

38.2.5 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()*

38.2.6 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()*

38.2.7 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()*

38.2.8 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()*

38.2.9 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

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()*

38.2.10 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).

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).

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_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **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_18TRANSFERS.

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_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **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_18TRANSFERS.
- **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_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **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_18TRANSFERS.

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_AF1
  - TIM_DMABASE_AF2
  - TIM_DMABASE_TISEL
- **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_18TRANSFERS.
- **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
- **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

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_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:**

38.3 TIM Firmware driver defines

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

38.3.1 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_LOCKUP

Enables and locks the LOCKUP (Hardfault) output of CortexM0 with Break Input of TIM1/16/17

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 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 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_DCR

TIM_DMABASE_DMAR

TIM_DMABASE_CCMR3

TIM_DMABASE_CCR5

TIM_DMABASE_CCR6

TIM_DMABASE_AF1

TIM_DMABASE_AF2

TIM_DMABASE_TISEL

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 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 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

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

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

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

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

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

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 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 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 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 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 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_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

39 HAL TIM Extension Driver

39.1 TIMEx Firmware driver registers structures

39.1.1 TIM_HallSensor_InitTypeDef

TIM_HallSensor_InitTypeDef is defined in the stm32c0xx_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

39.1.2 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32c0xx_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***

39.2 TIMEx Firmware driver API description

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

39.2.1 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

39.2.2 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. 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().

39.2.3 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()*

39.2.4 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()*

39.2.5 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.

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()*

39.2.6 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()*

39.2.7 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.

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()*

39.2.8 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()*

39.2.9 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()*

39.2.10 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

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

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

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

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
- **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

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

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

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

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

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
- **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

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_ITR7: Internal trigger 7 selected (*)
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **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_ITR7: Internal trigger 7 selected (*)
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **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_ITR7: Internal trigger 7 selected (*)
  - TIM_TS_NONE: No trigger is needed

  (*) Value not defined in all devices.
- **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 is connected to GPIO
  - 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

  For TIM2, the parameter can take one of the following values: (*)
  - TIM_TIM2_ETR_GPIO: TIM2_ETR is connected to GPIO
  - TIM_TIM2_ETR_LSE: TIM2_ETR is connected to LSE
  - TIM_TIM2_ETR_MCO: TIM2_ETR is connected to MCO
  - TIM_TIM2_ETR_MCO2: TIM2_ETR is connected to MCO2

  For TIM3, the parameter can take one of the following values:
  - TIM_TIM3_ETR_GPIO TIM3_ETR is connected to GPIO

  (*) Timer instance not available on 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
- **TISelection:** specifies the timer input source

Return values

- **HAL:** status

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_CommutCallback

Function name

void HAL_TIMEx_CommutCallback (TIM_HandleTypeDef * htim)

Function description

Commutation 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

Commutation 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

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

Break2 detection 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

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:**

39.3 TIMEx Firmware driver defines

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

39.3.1 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 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 Remapping

TIM_TIM1_ETR_GPIO

TIM1_ETR is connected to GPIO

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_TIM2_ETR_GPIO

TIM2_ETR is connected to GPIO

TIM_TIM2_ETR_LSE

TIM2_ETR is connected to LSE

TIM_TIM2_ETR_MCO

TIM2_ETR is connected to MCO

TIM_TIM2_ETR_MCO2

TIM2_ETR is connected to MCO2

TIM_TIM3_ETR_GPIO

TIM3_ETR is connected to GPIO

TIM Extended Timer input selection

TIM_TIM14_TI1_GPIO

TIM14_TI1 is connected to GPIO

TIM_TIM14_TI1_RTC

TIM14_TI1 is connected to RTC clock

TIM_TIM14_TI1_HSE_32

TIM14_TI1 is connected to HSE div 32

TIM_TIM14_TI1_MCO

TIM14_TI1 is connected to MCO

TIM_TIM14_TI1_MCO2

TIM14_TI1 is connected to MCO2

TIM_TIM15_TI1_GPIO

TIM15_TI1 is connected to GPIO

TIM_TIM15_TI1_TIM2_CH1

TIM15_TI1 is connected to TIM2 CH1

TIM_TIM15_TI1_TIM3_CH1

TIM15_TI1 is connected to TIM3 CH1

TIM_TIM15_TI2_GPIO

TIM15_TI2 is connected to GPIO

TIM_TIM15_TI2_TIM2_CH2

TIM15_TI2 is connected to TIM2 CH2

TIM_TIM15_TI2_TIM3_CH2

TIM15_TI2 is connected to TIM3 CH2

TIM_TIM16_TI1_GPIO

TIM16_TI1 is connected to GPIO

TIM_TIM16_TI1_LSI

TIM16_TI1 is connected to LSI

TIM_TIM16_TI1_LSE

TIM16_TI1 is connected to LSE

TIM_TIM16_TI1_MCO2

TIM16_TI1 is connected to MCO2

TIM_TIM17_TI1_GPIO

TIM17_TI1 is connected to GPIO

TIM_TIM17_TI1_HSE_32

TIM17_TI1 is connected to HSE div 32

TIM_TIM17_TI1_MCO

TIM17_TI1 is connected to MCO

TIM_TIM17_TI1_MCO2

TIM17_TI1 is connected to MCO2

40 HAL UART Generic Driver

40.1 UART Firmware driver registers structures

40.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32c0xx_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:
  - 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***.

40.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32c0xx_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***.

40.1.3 __UART_HandleTypeDef

__UART_HandleTypeDef is defined in the stm32c0xx_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(* WakeupCallback***
- ***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::WakeupCallback)(struct __UART_HandleTypeDef *huart)***
  UART Wakeup 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

40.2 UART Firmware driver API description

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

40.2.1 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.

40.2.2 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.

40.2.3 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()***

40.2.4 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()*

40.2.5 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()*

40.2.6 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()*

40.2.7 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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pData.

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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pData.

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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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 0x0FFFFFFF.

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.

40.3 UART Firmware driver defines

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

40.3.1 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
  - UART_CLEAR_WUF Wake Up from stop mode 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_WUF Wake up from stop mode 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_WUF Wakeup from stop mode 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_WUF Wakeup from stop mode 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_WUF Wakeup from stop mode 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_WUF Wakeup from stop mode 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
  - UART_CLEAR_WUF Wake Up from stop mode 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_WUF

UART wake-up from stop mode 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_WUF

UART wake-up from stop mode 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_WUF

Wake Up from stop mode 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_STARTBIT

UART wake-up on start bit

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

41 HAL UART Extension Driver

41.1 UARTEx Firmware driver registers structures

41.1.1 UART_WakeUpTypeDef

UART_WakeUpTypeDef is defined in the stm32c0xx_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).

41.2 UARTEx Firmware driver API description

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

41.2.1 UART peripheral extended features

41.2.2 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()*

41.2.3 IO operation functions

This section contains the following APIs:

- *HAL_UARTEx_WakeupCallback()*
- *HAL_UARTEx_RxFifoFullCallback()*
- *HAL_UARTEx_TxFifoEmptyCallback()*

41.2.4 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()***

41.2.5 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

UART wakeup from Stop mode callback.

Parameters

- **huart:** UART handle.

Return values

- **None:**

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_STARTBIT
  - 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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using uint16_t pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pData.

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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using uint16_t pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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;

41.3 UARTEx Firmware driver defines

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

41.3.1 UARTEx

UARTEx

UARTEx FIFO mode

UART_FIFOMODE_DISABLE

FIFO mode disable

UART_FIFOMODE_ENABLE

FIFO mode enable

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

42 HAL USART Generic Driver

42.1 USART Firmware driver registers structures

42.1.1 USART_InitTypeDef

USART_InitTypeDef is defined in the stm32c0xx_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***.

42.1.2 __USART_HandleTypeDef

__USART_HandleTypeDef is defined in the stm32c0xx_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 USART 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

42.2 USART Firmware driver API description

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

42.2.1 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.

42.2.2 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.

42.2.3 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in synchronous SPI master/slave mode.

- For the synchronous SPI 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 SPI 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()***

42.2.4 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous SPI data transfers.

The USART Synchronous SPI supports master and slave modes (SCLK as output or input).

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()*

42.2.5 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()*

42.2.6 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffers containing data to be sent/received, should be aligned on a half word frontier (16 bits) (as sent/received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pTxData and 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffers containing data to be sent/received, should be aligned on a half word frontier (16 bits) (as sent/received data will be handled using u16 pointer cast). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pTxData and 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer containing data to be sent, should be aligned on a half word frontier (16 bits) (as sent data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffer for storing data to be received, should be aligned on a half word frontier (16 bits) (as received data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for 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.
- When USART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01), address of user data buffers containing data to be sent/received, should be aligned on a half word frontier (16 bits) (as sent/received data will be handled by DMA from halfword frontier). Depending on compilation chain, use of specific alignment compilation directives or pragmas might be required to ensure proper alignment for pTxData and 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

42.3 USART Firmware driver defines

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

42.3.1 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

43 HAL USART Extension Driver

43.1 USARTEx Firmware driver API description

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

43.1.1 USART peripheral extended features

43.1.2 IO operation functions

This section contains the following APIs:

- *HAL_USARTEx_RxFifoFullCallback()*
- *HAL_USARTEx_TxFifoEmptyCallback()*

43.1.3 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()*

43.1.4 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

43.2 USARTEx Firmware driver defines

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

43.2.1 USARTEx

USARTEx

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

44 HAL WWDG Generic Driver

44.1 WWDG Firmware driver registers structures

44.1.1 WWDG_InitTypeDef

WWDG_InitTypeDef is defined in the stm32c0xx_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***

44.1.2 __WWDG_HandleTypeDef

__WWDG_HandleTypeDef is defined in the stm32c0xx_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

44.2 WWDG Firmware driver API description

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

44.2.1 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 32MHz (PCLK1) with zero prescaler: max timeout before reset: approximately 41.79us
  - Counter max (T[5;0] = 0x3F) at 32MHz (PCLK1) with prescaler dividing by 128: max timeout before reset: approximately 342.38ms

44.2.2 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

44.2.3 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()***

44.2.4 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()***

44.2.5 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:**

44.3 WWDG Firmware driver defines

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

44.3.1 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

45 LL ADC Generic Driver

45.1 ADC Firmware driver registers structures

45.1.1 LL_ADC_CommonInitTypeDef

LL_ADC_CommonInitTypeDef is defined in the stm32c0xx_ll_adc.h

Data Fields

- ***uint32_t CommonClock***

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()**.

45.1.2 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32c0xx_ll_adc.h

Data Fields

- ***uint32_t Clock***
- ***uint32_t Resolution***
- ***uint32_t DataAlignment***
- ***uint32_t LowPowerMode***

Field Documentation

- ***uint32_t LL_ADC_InitTypeDef::Clock***
  Set ADC instance clock source and prescaler. This parameter can be a value of ***ADC_LL_EC_CLOCK_SOURCE***
  Note:
  - On this STM32 series, this parameter has some clock ratio constraints: ADC clock synchronous (from PCLK) with prescaler 1 must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). This feature can be modified afterwards using unitary function **LL_ADC_SetClock()**. For more details, refer to description of this function.
- ***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::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()**.

45.1.3 LL_ADC_REG_InitTypeDef

LL_ADC_REG_InitTypeDef is defined in the stm32c0xx_ll_adc.h

Data Fields

- ***uint32_t TriggerSource***
- ***uint32_t SequencerLength***
- ***uint32_t SequencerDiscont***
- ***uint32_t ContinuousMode***
- ***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 series 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.
  Note:
  – This parameter has an effect only if group regular sequencer is set to mode "fully configurable". Refer to function **LL_ADC_REG_SetSequencerConfigurable()**. 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 (depending on the sequencer mode: scan length of 2 ranks or more, or several ADC channels enabled in group regular sequencer. Refer to function **LL_ADC_REG_SetSequencerConfigurable()**). 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::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()**.

45.2 ADC Firmware driver API description

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

45.2.1 Detailed description of functions

LL_ADC_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_ADC_DMA_GetRegAddr (const ADC_TypeDef * ADCx, uint32_t Register)

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
- **Register:** This parameter can be one of the following values:
  - LL_ADC_DMA_REG_REGULAR_DATA

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 DATA 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 (1)
  - LL_ADC_CLOCK_ASYNC_DIV2 (1)
  - LL_ADC_CLOCK_ASYNC_DIV4 (1)
  - LL_ADC_CLOCK_ASYNC_DIV6 (1)
  - LL_ADC_CLOCK_ASYNC_DIV8 (1)
  - LL_ADC_CLOCK_ASYNC_DIV10 (1)
  - LL_ADC_CLOCK_ASYNC_DIV12 (1)
  - LL_ADC_CLOCK_ASYNC_DIV16 (1)
  - LL_ADC_CLOCK_ASYNC_DIV32 (1)
  - LL_ADC_CLOCK_ASYNC_DIV64 (1)
  - LL_ADC_CLOCK_ASYNC_DIV128 (1)
  - LL_ADC_CLOCK_ASYNC_DIV256 (1)

  (1) ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous. (refer to function LL_ADC_SetClock()).

Return values

- **None:**

Notes

- 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 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 (1)
  - LL_ADC_CLOCK_ASYNC_DIV2 (1)
  - LL_ADC_CLOCK_ASYNC_DIV4 (1)
  - LL_ADC_CLOCK_ASYNC_DIV6 (1)
  - LL_ADC_CLOCK_ASYNC_DIV8 (1)
  - LL_ADC_CLOCK_ASYNC_DIV10 (1)
  - LL_ADC_CLOCK_ASYNC_DIV12 (1)
  - LL_ADC_CLOCK_ASYNC_DIV16 (1)
  - LL_ADC_CLOCK_ASYNC_DIV32 (1)
  - LL_ADC_CLOCK_ASYNC_DIV64 (1)
  - LL_ADC_CLOCK_ASYNC_DIV128 (1)
  - LL_ADC_CLOCK_ASYNC_DIV256 (1)

  (1) ADC common clock asynchronous prescaler is applied to each ADC instance if the corresponding ADC instance clock is set to clock source asynchronous. (refer to function LL_ADC_SetClock()).

Reference Manual to LL API cross reference:

- CCR PRESC LL_ADC_GetCommonClock

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

Return values

- **None:**

Notes

- One or several values can be selected. Example: (LL_ADC_PATH_INTERNAL_VREFINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR) The values not selected are removed from configuration.
- 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, LL_ADC_DELAY_TEMPSENSOR_BUFFER_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 TSEN LL_ADC_SetCommonPathInternalCh
- CCR VBATEN LL_ADC_SetCommonPathInternalCh

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

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, LL_ADC_DELAY_TEMPSENSOR_BUFFER_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 TSEN 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

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 TSEN LL_ADC_SetCommonPathInternalChRem
-

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

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 TSEN LL_ADC_GetCommonPathInternalCh
- CCR VBATEN LL_ADC_GetCommonPathInternalCh

LL_ADC_SetClock

Function name

__STATIC_INLINE void LL_ADC_SetClock (ADC_TypeDef * ADCx, uint32_t ClockSource)

Function description

Set ADC instance clock source and prescaler.

Parameters

- **ADCx:** ADC instance
- **ClockSource:** This parameter can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1 (2)
  - LL_ADC_CLOCK_ASYNC (1)

  (1) Asynchronous clock prescaler can be configured using function LL_ADC_SetCommonClock().
- (2) Caution: This parameter has some clock ratio constraints: This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). Refer to reference manual.

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:

- CFGR2 CKMODE LL_ADC_SetClock

LL_ADC_GetClock

Function name

__STATIC_INLINE uint32_t LL_ADC_GetClock (const ADC_TypeDef * ADCx)

Function description

Get ADC instance clock source and prescaler.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CLOCK_SYNC_PCLK_DIV4
  - LL_ADC_CLOCK_SYNC_PCLK_DIV2
  - LL_ADC_CLOCK_SYNC_PCLK_DIV1 (2)
  - LL_ADC_CLOCK_ASYNC (1)

  (1) Asynchronous clock prescaler can be retrieved using function LL_ADC_GetCommonClock().
- (2) Caution: This parameter has some clock ratio constraints: This configuration must be enabled only if PCLK has a 50% duty clock cycle (APB prescaler configured inside the RCC must be bypassed and the system clock must by 50% duty cycle). Refer to reference manual.

Reference Manual to LL API cross reference:

- CFGR2 CKMODE LL_ADC_GetClock

LL_ADC_SetCalibrationFactor

Function name

__STATIC_INLINE void LL_ADC_SetCalibrationFactor (ADC_TypeDef * ADCx, 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
- **CalibrationFactor:** Value between Min_Data=0x00 and Max_Data=0x7F

Return values

- **None:**

Notes

- This function is intended to set calibration parameters without having to perform a new calibration using LL_ADC_StartCalibration().
- 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:

- CALFACT CALFACT LL_ADC_SetCalibrationFactor

LL_ADC_GetCalibrationFactor

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibrationFactor (const ADC_TypeDef * ADCx)

Function description

Get ADC calibration factor in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7F

Notes

- Calibration factors are set by hardware after performing a calibration run using function LL_ADC_StartCalibration().

Reference Manual to LL API cross reference:

- CALFACT CALFACT LL_ADC_GetCalibrationFactor

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_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

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:

- CFGR1 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_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

Reference Manual to LL API cross reference:

- CFGR1 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.

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
  - LL_ADC_LP_AUTOPOWEROFF
  - LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF

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) 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.

Reference Manual to LL API cross reference:

- CFGR1 WAIT LL_ADC_SetLowPowerMode
- CFGR1 AUTOFF 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
  - LL_ADC_LP_AUTOPOWEROFF
  - LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF

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) 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:

- CFGR1 WAIT LL_ADC_GetLowPowerMode
- CFGR1 AUTOFF LL_ADC_GetLowPowerMode

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.

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_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_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_3CYCLES_5
  - LL_ADC_SAMPLINGTIME_7CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_19CYCLES_5
  - LL_ADC_SAMPLINGTIME_39CYCLES_5
  - LL_ADC_SAMPLINGTIME_79CYCLES_5
  - LL_ADC_SAMPLINGTIME_160CYCLES_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.

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
- **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_ADC_SAMPLINGTIME_1CYCLE_5
  - LL_ADC_SAMPLINGTIME_3CYCLES_5
  - LL_ADC_SAMPLINGTIME_7CYCLES_5
  - LL_ADC_SAMPLINGTIME_12CYCLES_5
  - LL_ADC_SAMPLINGTIME_19CYCLES_5
  - LL_ADC_SAMPLINGTIME_39CYCLES_5
  - LL_ADC_SAMPLINGTIME_79CYCLES_5
  - LL_ADC_SAMPLINGTIME_160CYCLES_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_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH4
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11 (1) ADC external trigger available on STM32C071xx device only.

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().
- On this STM32 series, ADC trigger frequency mode must be set in function of frequency of ADC group regular conversion trigger. Refer to description of function LL_ADC_SetTriggerFrequencyMode().
- 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.

Reference Manual to LL API cross reference:

- CFGR1 EXTSEL LL_ADC_REG_SetTriggerSource
- CFGR1 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_TRGO2
  - LL_ADC_REG_TRIG_EXT_TIM1_CH4
  - LL_ADC_REG_TRIG_EXT_TIM2_TRGO (1)
  - LL_ADC_REG_TRIG_EXT_TIM3_TRGO
  - LL_ADC_REG_TRIG_EXT_EXTI_LINE11 (1) ADC external trigger available on STM32C071xx device only.

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:

- CFGR1 EXTSEL LL_ADC_REG_GetTriggerSource
- CFGR1 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:

- CFGR1 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:

- CFGR1 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:

- CFGR1 EXTEN LL_ADC_REG_GetTriggerEdge

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, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled.

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().

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

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()". To set scan direction differently, refer to function LL_ADC_REG_SetSequencerScanDirection().
- 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().
- On this STM32 series, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- 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:

- CHSELR SQ1 LL_ADC_REG_SetSequencerLength
- CHSELR SQ2 LL_ADC_REG_SetSequencerLength
- CHSELR SQ3 LL_ADC_REG_SetSequencerLength
- CHSELR SQ4 LL_ADC_REG_SetSequencerLength
- CHSELR SQ5 LL_ADC_REG_SetSequencerLength
- CHSELR SQ6 LL_ADC_REG_SetSequencerLength
- CHSELR SQ7 LL_ADC_REG_SetSequencerLength
- CHSELR SQ8 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

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()". To set scan direction differently, refer to function LL_ADC_REG_SetSequencerScanDirection().
- 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().
- Sequencer disabled is equivalent to sequencer of 1 rank: ADC conversion on only 1 channel.

Reference Manual to LL API cross reference:

- CHSELR SQ1 LL_ADC_REG_GetSequencerLength
- CHSELR SQ2 LL_ADC_REG_GetSequencerLength
- CHSELR SQ3 LL_ADC_REG_GetSequencerLength
- CHSELR SQ4 LL_ADC_REG_GetSequencerLength
- CHSELR SQ5 LL_ADC_REG_GetSequencerLength
- CHSELR SQ6 LL_ADC_REG_GetSequencerLength
- CHSELR SQ7 LL_ADC_REG_GetSequencerLength
- CHSELR SQ8 LL_ADC_REG_GetSequencerLength

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 series, this setting is not available and the default scan direction is forward.
- On this STM32 series, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled.

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_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

Return values

- **None:**

Notes

- 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.

Reference Manual to LL API cross reference:

- CFGR1 DISCEN 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

Reference Manual to LL API cross reference:

- CFGR1 DISCEN 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
- **Channel:** This parameter 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

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, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- 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:

- CHSELR SQ1 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ2 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ3 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ4 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ5 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ6 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ7 LL_ADC_REG_SetSequencerRanks
- CHSELR SQ8 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

Return values

- **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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

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:

- CHSELR SQ1 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ2 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ3 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ4 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ5 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ6 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ7 LL_ADC_REG_GetSequencerRanks
- CHSELR SQ8 LL_ADC_REG_GetSequencerRanks

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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

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().
- On this STM32 series, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- 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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

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().
- On this STM32 series, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- 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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

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().
- On this STM32 series, after modifying sequencer (functions LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks(), ...) it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, some actions may be ignored. Refer to description of LL_ADC_IsActiveFlag_CCRDY for more details.
- 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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

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_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.

Reference Manual to LL API cross reference:

- CFGR1 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:

- CFGR1 CONT LL_ADC_REG_GetContinuousMode

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.

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.

Reference Manual to LL API cross reference:

- CFGR1 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:

- CFGR1 OVRMOD LL_ADC_REG_GetOverrun

LL_ADC_SetChannelSamplingTime

Function name

__STATIC_INLINE void LL_ADC_SetChannelSamplingTime (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t SamplingTimeY)

Function description

Set sampling time of the selected ADC channel Unit: ADC clock cycles.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be a combination 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.
- **SamplingTimeY:** This parameter can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_1
  - LL_ADC_SAMPLINGTIME_COMMON_2

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. Refer to reference manual for ADC processing time of this STM32 series.
- 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.

Reference Manual to LL API cross reference:

- SMPR SMPSEL0 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL1 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL2 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL3 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL4 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL5 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL6 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL7 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL8 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL9 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL10 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL11 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL12 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL13 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL14 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL15 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL16 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL17 LL_ADC_SetChannelSamplingTime
- SMPR SMPSEL18 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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)

  (1) On STM32C0, 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().
- (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(). (3) ADC channels available on STM32C031xx device only.

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLINGTIME_COMMON_1
  - LL_ADC_SAMPLINGTIME_COMMON_2

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. Refer to reference manual for ADC processing time of this STM32 series.

Reference Manual to LL API cross reference:

- SMPR SMPSEL0 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL1 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL2 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL3 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL4 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL5 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL6 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL7 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL8 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL9 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL10 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL11 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL12 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL13 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL14 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL15 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL16 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL17 LL_ADC_GetChannelSamplingTime
- SMPR SMPSEL18 LL_ADC_GetChannelSamplingTime

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 or all channels, on ADC group regular.

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
  - LL_ADC_AWD_CHANNEL_0_REG
  - LL_ADC_AWD_CHANNEL_1_REG
  - LL_ADC_AWD_CHANNEL_2_REG
  - LL_ADC_AWD_CHANNEL_3_REG
  - LL_ADC_AWD_CHANNEL_4_REG
  - LL_ADC_AWD_CHANNEL_5_REG
  - LL_ADC_AWD_CHANNEL_6_REG
  - LL_ADC_AWD_CHANNEL_7_REG
  - LL_ADC_AWD_CHANNEL_8_REG
  - LL_ADC_AWD_CHANNEL_9_REG
  - LL_ADC_AWD_CHANNEL_10_REG
  - LL_ADC_AWD_CHANNEL_11_REG
  - LL_ADC_AWD_CHANNEL_12_REG
  - LL_ADC_AWD_CHANNEL_13_REG
  - LL_ADC_AWD_CHANNEL_14_REG
  - LL_ADC_AWD_CHANNEL_15_REG
  - LL_ADC_AWD_CHANNEL_16_REG
  - LL_ADC_AWD_CHANNEL_17_REG
  - LL_ADC_AWD_CHANNEL_18_REG (1)
  - LL_ADC_AWD_CHANNEL_19_REG (1)
  - LL_ADC_AWD_CHANNEL_20_REG (1)
  - LL_ADC_AWD_CHANNEL_21_REG (1)
  - LL_ADC_AWD_CHANNEL_22_REG (1)
  - LL_ADC_AWD_CH_VREFINT_REG
  - LL_ADC_AWD_CH_TEMPSENSOR_REG
  - LL_ADC_AWD_CH_VDDA_REG
  - LL_ADC_AWD_CH_VSSA_REG (1) On STM32C0, parameter not available on all devices: only on STM32C031xx.

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 is only 1 kind 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).
- On this STM32 series, setting of this feature is conditioned to ADC state: ADC must be disabled.

Reference Manual to LL API cross reference:

- CFGR1 AWD1CH LL_ADC_SetAnalogWDMonitChannels
- CFGR1 AWD1SGL LL_ADC_SetAnalogWDMonitChannels
- CFGR1 AWD1EN 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
  - LL_ADC_AWD_CHANNEL_0_REG
  - LL_ADC_AWD_CHANNEL_1_REG
  - LL_ADC_AWD_CHANNEL_2_REG
  - LL_ADC_AWD_CHANNEL_3_REG
  - LL_ADC_AWD_CHANNEL_4_REG
  - LL_ADC_AWD_CHANNEL_5_REG
  - LL_ADC_AWD_CHANNEL_6_REG
  - LL_ADC_AWD_CHANNEL_7_REG
  - LL_ADC_AWD_CHANNEL_8_REG
  - LL_ADC_AWD_CHANNEL_9_REG
  - LL_ADC_AWD_CHANNEL_10_REG
  - LL_ADC_AWD_CHANNEL_11_REG
  - LL_ADC_AWD_CHANNEL_12_REG
  - LL_ADC_AWD_CHANNEL_13_REG
  - LL_ADC_AWD_CHANNEL_14_REG
  - LL_ADC_AWD_CHANNEL_15_REG
  - LL_ADC_AWD_CHANNEL_16_REG
  - LL_ADC_AWD_CHANNEL_17_REG
  - LL_ADC_AWD_CHANNEL_18_REG (1)
  - LL_ADC_AWD_CHANNEL_19_REG (1)
  - LL_ADC_AWD_CHANNEL_20_REG (1)
  - LL_ADC_AWD_CHANNEL_21_REG (1)
  - LL_ADC_AWD_CHANNEL_22_REG (1)
  - LL_ADC_AWD_CH_VREFINT_REG
  - LL_ADC_AWD_CH_TEMPSENSOR_REG
  - LL_ADC_AWD_CH_VDDA_REG
  - LL_ADC_AWD_CH_VSSA_REG (1) On STM32C0, parameter not available on all devices: only on STM32C031xx.

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 is only 1 kind 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).
- 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 AWD1CH LL_ADC_GetAnalogWDMonitChannels
- CFGR1 AWD1SGL LL_ADC_GetAnalogWDMonitChannels
- CFGR1 AWD1EN LL_ADC_GetAnalogWDMonitChannels
- AWD2CR AWD2CH LL_ADC_GetAnalogWDMonitChannels
- AWD3CR AWD3CH LL_ADC_GetAnalogWDMonitChannels

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 is only 2 kind 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:

- AWD1TR HT1 LL_ADC_ConfigAnalogWDThresholds
- AWD2TR HT2 LL_ADC_ConfigAnalogWDThresholds
- AWD3TR HT3 LL_ADC_ConfigAnalogWDThresholds
- AWD1TR LT1 LL_ADC_ConfigAnalogWDThresholds
- AWD2TR LT2 LL_ADC_ConfigAnalogWDThresholds
- AWD3TR LT3 LL_ADC_ConfigAnalogWDThresholds

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

- If values of both thresholds high or low must be set, use function LL_ADC_ConfigAnalogWDThresholds().
- 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 is only 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 not conditioned to ADC state: ADC can be disabled, enabled with or without conversion on going on ADC group regular.

Reference Manual to LL API cross reference:

- AWD1TR HT1 LL_ADC_SetAnalogWDThresholds
- AWD2TR HT2 LL_ADC_SetAnalogWDThresholds
- AWD3TR HT3 LL_ADC_SetAnalogWDThresholds
- AWD1TR LT1 LL_ADC_SetAnalogWDThresholds
- AWD2TR LT2 LL_ADC_SetAnalogWDThresholds
- AWD3TR 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
  - LL_ADC_AWD_THRESHOLDS_HIGH_LOW

Return values

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

Notes

- If raw data with ADC thresholds high and low is retrieved, the data of each threshold high or low can be isolated using helper macro: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW().
- 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:

- AWD1TR HT1 LL_ADC_GetAnalogWDThresholds
- AWD2TR HT2 LL_ADC_GetAnalogWDThresholds
- AWD3TR HT3 LL_ADC_GetAnalogWDThresholds
- AWD1TR LT1 LL_ADC_GetAnalogWDThresholds
- AWD2TR LT2 LL_ADC_GetAnalogWDThresholds
- AWD3TR LT3 LL_ADC_GetAnalogWDThresholds

LL_ADC_SetOverSamplingScope

Function name

__STATIC_INLINE void LL_ADC_SetOverSamplingScope (ADC_TypeDef * ADCx, uint32_t OvsScope)

Function description

Set ADC oversampling scope.

Parameters

- **ADCx:** ADC instance
- **OvsScope:** This parameter can be one of the following values:
  - LL_ADC_OVS_DISABLE
  - LL_ADC_OVS_GRP_REGULAR_CONTINUED

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:

- CFGR2 OVSE 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.

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

Reference Manual to LL API cross reference:

- CFGR2 OVSE 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.

Reference Manual to LL API cross reference:

- CFGR2 TOVS 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 TOVS 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.

Parameters

- **ADCx:** ADC instance
- **Ratio:** 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

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.

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.

Parameters

- **ADCx:** ADC instance

Return values

- **Ratio:** 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

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.

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

Reference Manual to LL API cross reference:

- CFGR2 OVSS LL_ADC_GetOverSamplingShift

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, there are three possibilities to enable the voltage regulator: by enabling it manually using function LL_ADC_EnableInternalRegulator().by launching a calibration using function LL_ADC_StartCalibration().by enabling the ADC using function LL_ADC_Enable().
- 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 group regular.

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)

Function description

Start ADC calibration in the mode single-ended or differential (for devices with differential mode available).

Parameters

- **ADCx:** ADC instance

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.
- In case of usage of ADC with DMA transfer: On this STM32 series, ADC DMA transfer request should be disabled during calibration: Calibration factor is available in data register and also transferred by DMA. To not insert ADC calibration factor among ADC conversion data in array variable, DMA transfer must be disabled during calibration. (DMA transfer setting backup and disable before calibration, DMA transfer setting restore after calibration. Refer to functions LL_ADC_REG_GetDMATransfer(), LL_ADC_REG_SetDMATransfer()).
- In case of usage of feature auto power-off: This mode must be disabled during calibration Refer to function LL_ADC_SetLowPowerMode().
- 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

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 DATA LL_ADC_REG_ReadConversionData32

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 DATA 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 DATA 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 DATA LL_ADC_REG_ReadConversionData8

LL_ADC_REG_ReadConversionData6

Function name

__STATIC_INLINE uint8_t LL_ADC_REG_ReadConversionData6 (const ADC_TypeDef * ADCx)

Function description

Get ADC group regular conversion data, range fit for ADC resolution 6 bits.

Parameters

- **ADCx:** ADC instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x3F

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 DATA LL_ADC_REG_ReadConversionData6

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_CCRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_CCRDY (const ADC_TypeDef * ADCx)

Function description

Get flag ADC channel configuration ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Notes

- On this STM32 series, after modifying sequencer it is mandatory to wait for the assertion of CCRDY flag using LL_ADC_IsActiveFlag_CCRDY(). Otherwise, performing some actions (configuration update, ADC conversion start, ...) will be ignored. Functions requiring wait for CCRDY flag are: LL_ADC_REG_SetSequencerLength() LL_ADC_REG_SetSequencerRanks() LL_ADC_REG_SetSequencerChannels() LL_ADC_REG_SetSequencerChAdd() LL_ADC_REG_SetSequencerChRem() LL_ADC_REG_SetSequencerScanDirection() LL_ADC_REG_SetSequencerConfigurable()
- Duration of ADC channel configuration ready: CCRDY handshake requires 1APB + 2 ADC + 3 APB cycles after the channel configuration has been changed.

Reference Manual to LL API cross reference:

- ISR CCRDY LL_ADC_IsActiveFlag_CCRDY

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_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_IsActiveFlag_EOCAL

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOCAL (const ADC_TypeDef * ADCx)

Function description

Get flag ADC end of calibration.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR EOCAL LL_ADC_IsActiveFlag_EOCAL

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_CCRDY

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_CCRDY (ADC_TypeDef * ADCx)

Function description

Clear flag ADC channel configuration ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR CCRDY LL_ADC_ClearFlag_CCRDY

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_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_ClearFlag_EOCAL

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOCAL (ADC_TypeDef * ADCx)

Function description

Clear flag ADC end of calibration.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR EOCAL LL_ADC_ClearFlag_EOCAL

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_CCRDY

Function name

__STATIC_INLINE void LL_ADC_EnableIT_CCRDY (ADC_TypeDef * ADCx)

Function description

Enable interruption ADC channel configuration ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER ADRDYIE LL_ADC_EnableIT_CCRDY

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_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_CCRDY

Function name

__STATIC_INLINE void LL_ADC_DisableIT_CCRDY (ADC_TypeDef * ADCx)

Function description

Disable interruption ADC channel configuration ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CCRDYIE LL_ADC_DisableIT_CCRDY

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_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_CCRDY

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_CCRDY (const ADC_TypeDef * ADCx)

Function description

Get state of interruption ADC channel configuration ready.

Parameters

- **ADCx:** ADC instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- IER CCRDYIE LL_ADC_IsEnabledIT_CCRDY

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_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 (const ADC_Common_TypeDef * ADCxy_COMMON)

Function description

De-initialize registers of all ADC instances belonging to the same ADC common instance to their default reset values.

Parameters

- **ADCxy_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.

LL_ADC_CommonInit

Function name

ErrorStatus LL_ADC_CommonInit (ADC_Common_TypeDef * ADCxy_COMMON, const LL_ADC_CommonInitTypeDef * ADC_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

- **ADCxy_COMMON:** ADC common instance (can be set directly from CMSIS definition or by using helper macro __LL_ADC_COMMON_INSTANCE())
- **ADC_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 * ADC_CommonInitStruct)

Function description

Set each LL_ADC_CommonInitTypeDef field to default value.

Parameters

- **ADC_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 * ADCx)

Function description

De-initialize registers of the selected ADC instance to their default reset values.

Parameters

- **ADCx:** 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().
- If this functions returns error status, it means that ADC instance is in an unknown state. In this case, perform a hard reset using high level clock source RCC ADC reset. Refer to function LL_ADC_CommonDeInit().

LL_ADC_Init

Function name

ErrorStatus LL_ADC_Init (ADC_TypeDef * ADCx, const LL_ADC_InitTypeDef * ADC_InitStruct)

Function description

Initialize some features of ADC instance.

Parameters

- **ADCx:** ADC instance
- **ADC_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. 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 sequencer: Depending on the sequencer mode (refer to function LL_ADC_REG_SetSequencerConfigurable()): map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks();map channel on rank corresponding to channel number. Refer to function LL_ADC_REG_SetSequencerChannels(); Set ADC channel sampling time Refer to function LL_ADC_SetSamplingTimeCommonChannels(); Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_StructInit

Function name

void LL_ADC_StructInit (LL_ADC_InitTypeDef * ADC_InitStruct)

Function description

Set each LL_ADC_InitTypeDef field to default value.

Parameters

- **ADC_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 * ADCx, const LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)

Function description

Initialize some features of ADC group regular.

Parameters

- **ADCx:** ADC instance
- **ADC_REG_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 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.
- Before using this function, ADC group regular sequencer must be configured: refer to function LL_ADC_REG_SetSequencerConfigurable().
- 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 sequencer: Depending on the sequencer mode (refer to function LL_ADC_REG_SetSequencerConfigurable()): map channel on the selected sequencer rank. Refer to function LL_ADC_REG_SetSequencerRanks();map channel on rank corresponding to channel number. Refer to function LL_ADC_REG_SetSequencerChannels(); Set ADC channel sampling time Refer to function LL_ADC_SetSamplingTimeCommonChannels(); Refer to function LL_ADC_SetChannelSamplingTime();

LL_ADC_REG_StructInit

Function name

void LL_ADC_REG_StructInit (LL_ADC_REG_InitTypeDef * ADC_REG_InitStruct)

Function description

Set each LL_ADC_REG_InitTypeDef field to default value.

Parameters

- **ADC_REG_InitStruct:** Pointer to a LL_ADC_REG_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

45.3 ADC Firmware driver defines

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

45.3.1 ADC

ADC

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_CHANNEL_0_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN0, converted by group regular only

LL_ADC_AWD_CHANNEL_1_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN1, converted by group regular only

LL_ADC_AWD_CHANNEL_2_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN2, converted by group regular only

LL_ADC_AWD_CHANNEL_3_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN3, converted by group regular only

LL_ADC_AWD_CHANNEL_4_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN4, converted by group regular only

LL_ADC_AWD_CHANNEL_5_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN5, converted by group regular only

LL_ADC_AWD_CHANNEL_6_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN6, converted by group regular only

LL_ADC_AWD_CHANNEL_7_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN7, converted by group regular only

LL_ADC_AWD_CHANNEL_8_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN8, converted by group regular only

LL_ADC_AWD_CHANNEL_9_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN9, converted by group regular only

LL_ADC_AWD_CHANNEL_10_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN10, converted by group regular only

LL_ADC_AWD_CHANNEL_11_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN11, converted by group regular only

LL_ADC_AWD_CHANNEL_12_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN12, converted by group regular only

LL_ADC_AWD_CHANNEL_13_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN13, converted by group regular only

LL_ADC_AWD_CHANNEL_14_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN14, converted by group regular only

LL_ADC_AWD_CHANNEL_15_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN15, converted by group regular only

LL_ADC_AWD_CHANNEL_16_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN16, converted by group regular only

LL_ADC_AWD_CHANNEL_17_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN17, converted by group regular only

LL_ADC_AWD_CHANNEL_18_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN18, converted by group regular only

LL_ADC_AWD_CHANNEL_19_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN19, converted by group regular only

LL_ADC_AWD_CHANNEL_20_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN20, converted by group regular only

LL_ADC_AWD_CHANNEL_21_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN21, converted by group regular only

LL_ADC_AWD_CHANNEL_22_REG

ADC analog watchdog monitoring of ADC channel ADCx_IN22, converted by group regular only

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_TEMPSENSOR_REG

ADC analog watchdog monitoring of ADC internal channel connected to Temperature sensor, converted by group regular only

LL_ADC_AWD_CH_VSSA_REG

ADC analog watchdog monitoring of ADC internal channel connected to VSSA : internal analog supply ground, converted by group regular only

LL_ADC_AWD_CH_VDDA_REG

ADC analog watchdog monitoring of ADC internal channel connected to VDDA : internal analog power supply, converted by group regular only

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

LL_ADC_AWD_THRESHOLDS_HIGH_LOW

ADC analog watchdog both thresholds high and low concatenated into the same data

ADC instance - Channel number

LL_ADC_CHANNEL_0

ADC channel ADCx_IN0

LL_ADC_CHANNEL_1

ADC channel ADCx_IN1

LL_ADC_CHANNEL_2

ADC channel ADCx_IN2

LL_ADC_CHANNEL_3

ADC channel ADCx_IN3

LL_ADC_CHANNEL_4

ADC channel ADCx_IN4

LL_ADC_CHANNEL_5

ADC channel ADCx_IN5

LL_ADC_CHANNEL_6

ADC channel ADCx_IN6

LL_ADC_CHANNEL_7

ADC channel ADCx_IN7

LL_ADC_CHANNEL_8

ADC channel ADCx_IN8

LL_ADC_CHANNEL_9

ADC channel ADCx_IN9

LL_ADC_CHANNEL_10

ADC channel ADCx_IN10

LL_ADC_CHANNEL_11

ADC channel ADCx_IN11

LL_ADC_CHANNEL_12

ADC channel ADCx_IN12

LL_ADC_CHANNEL_13

ADC channel ADCx_IN13

LL_ADC_CHANNEL_14

ADC channel ADCx_IN14

LL_ADC_CHANNEL_15

ADC channel ADCx_IN15

LL_ADC_CHANNEL_16

ADC channel ADCx_IN16

LL_ADC_CHANNEL_17

ADC channel ADCx_IN17

LL_ADC_CHANNEL_18

ADC channel ADCx_IN18

LL_ADC_CHANNEL_19

ADC channel ADCx_IN19

LL_ADC_CHANNEL_20

ADC channel ADCx_IN20

LL_ADC_CHANNEL_21

ADC channel ADCx_IN21

LL_ADC_CHANNEL_22

ADC channel ADCx_IN22

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_VDDA

ADC internal channel connected to VDDA.

LL_ADC_CHANNEL_VSSA

ADC internal channel connected to VSSA.

Channel - Sampling time

LL_ADC_SAMPLINGTIME_1CYCLE_5

Sampling time 1.5 ADC clock cycle

LL_ADC_SAMPLINGTIME_3CYCLES_5

Sampling time 3.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_7CYCLES_5

Sampling time 7.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_12CYCLES_5

Sampling time 12.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_19CYCLES_5

Sampling time 19.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_39CYCLES_5

Sampling time 39.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_79CYCLES_5

Sampling time 79.5 ADC clock cycles

LL_ADC_SAMPLINGTIME_160CYCLES_5

Sampling time 160.5 ADC clock cycles

ADC instance - Clock source

LL_ADC_CLOCK_SYNC_PCLK_DIV4

ADC synchronous clock derived from AHB clock divided by 4

LL_ADC_CLOCK_SYNC_PCLK_DIV2

ADC synchronous clock derived from AHB clock divided by 2

LL_ADC_CLOCK_SYNC_PCLK_DIV1

ADC synchronous clock derived from AHB clock not divided

LL_ADC_CLOCK_ASYNC

ADC asynchronous clock. Asynchronous clock prescaler can be configured using function

ADC common - Clock frequency mode

LL_ADC_CLOCK_FREQ_MODE_HIGH

LL ADC define maintained for legacy purpose

LL_ADC_CLOCK_FREQ_MODE_LOW

LL ADC define maintained for legacy purpose

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. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV4

ADC asynchronous clock with prescaler division by 4. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV6

ADC asynchronous clock with prescaler division by 6. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV8

ADC asynchronous clock with prescaler division by 8. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV10

ADC asynchronous clock with prescaler division by 10. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV12

ADC asynchronous clock with prescaler division by 12. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV16

ADC asynchronous clock with prescaler division by 16. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV32

ADC asynchronous clock with prescaler division by 32. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV64

ADC asynchronous clock with prescaler division by 64. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV128

ADC asynchronous clock with prescaler division by 128. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

LL_ADC_CLOCK_ASYNC_DIV256

ADC asynchronous clock with prescaler division by 256. Setting common to ADC instances of ADC common group, applied ADC instance wise to each instance clock set to clock source asynchronous (refer to function

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

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_CCRDY

ADC flag ADC channel configuration 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_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

ADC instance - Groups

LL_ADC_GROUP_REGULAR

ADC group regular (available on all STM32 devices)

Definitions of constants used by helper macro

LL_ADC_TEMPERATURE_CALC_ERROR

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 (starting from temperature sensor enable, refer to

LL_ADC_DELAY_TEMPSENSOR_BUFFER_STAB_US

Delay for temperature sensor buffer stabilization time (starting from ADC enable, refer to

LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES

Delay required between ADC end of calibration and ADC enable

ADC interruptions for configuration (interruption enable or disable)

LL_ADC_IT_ADRDY

ADC interruption ADC instance ready

LL_ADC_IT_CCRDY

ADC interruption channel configuration 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_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

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

LL_ADC_LP_AUTOPOWEROFF

ADC low power mode auto power-off: 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

LL_ADC_LP_AUTOWAIT_AUTOPOWEROFF

ADC low power modes auto wait and auto power-off combined. See description with function

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 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_4

ADC oversampling ratio of 4 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_8

ADC oversampling ratio of 8 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_16

ADC oversampling ratio of 16 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_32

ADC oversampling ratio of 32 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_64

ADC oversampling ratio of 64 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_128

ADC oversampling ratio of 128 (sum of conversions data computed to result as oversampling conversion data (before potential shift)

LL_ADC_OVS_RATIO_256

ADC oversampling ratio of 256 (sum of conversions data computed to result as 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. Literal suffix "continued" is kept for compatibility with other STM32 devices featuring ADC group injected, in this case other oversampling scope parameters are available.

Oversampling - Data right shift

LL_ADC_OVS_SHIFT_NONE

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

LL_ADC_OVS_SHIFT_RIGHT_1

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

LL_ADC_OVS_SHIFT_RIGHT_2

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

LL_ADC_OVS_SHIFT_RIGHT_3

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

LL_ADC_OVS_SHIFT_RIGHT_4

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

LL_ADC_OVS_SHIFT_RIGHT_5

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

LL_ADC_OVS_SHIFT_RIGHT_6

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

LL_ADC_OVS_SHIFT_RIGHT_7

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

LL_ADC_OVS_SHIFT_RIGHT_8

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

ADC registers compliant with specific purpose

LL_ADC_DMA_REG_REGULAR_DATA

ADC group regular - Continuous mode

LL_ADC_REG_CONV_SINGLE

ADC conversions performed in single mode: one conversion per trigger

LL_ADC_REG_CONV_CONTINUOUS

ADC conversions performed in continuous mode: after the first trigger, following conversions launched successively automatically

ADC group regular - DMA transfer of ADC conversion data

LL_ADC_REG_DMA_TRANSFER_NONE

ADC conversions are not transferred by DMA

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.

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 - 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

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 seq. rank 1

LL_ADC_REG_RANK_2

ADC group regular seq. rank 2

LL_ADC_REG_RANK_3

ADC group regular seq. rank 3

LL_ADC_REG_RANK_4

ADC group regular seq. rank 4

LL_ADC_REG_RANK_5

ADC group regular seq. rank 5

LL_ADC_REG_RANK_6

ADC group regular seq. rank 6

LL_ADC_REG_RANK_7

ADC group regular seq. rank 7

LL_ADC_REG_RANK_8

ADC group regular seq. rank 8

ADC group regular - Sequencer scan direction

LL_ADC_REG_SEQ_SCAN_DIR_FORWARD

On this STM32 series, parameter relevant only if sequencer set to mode not fully configurable, refer to function

LL_ADC_REG_SEQ_SCAN_DIR_BACKWARD

On this STM32 series, parameter relevant only if 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

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_TRGO2

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

LL_ADC_REG_TRIG_EXT_TIM1_CH4

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).

LL_ADC_REG_TRIG_EXT_TIM2_TRGO

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

ADC instance - Resolution

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

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

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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (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
  - LL_ADC_CHANNEL_20
  - LL_ADC_CHANNEL_21
  - LL_ADC_CHANNEL_22

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:
  - ADC_CHANNEL_VREFINT
  - ADC_CHANNEL_TEMPSENSOR
  - ADC_CHANNEL_VDDA
  - ADC_CHANNEL_VSSA

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:
  - 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 (1)
  - ADC_CHANNEL_9 (1)
  - ADC_CHANNEL_10
  - ADC_CHANNEL_11
  - ADC_CHANNEL_12
  - ADC_CHANNEL_13
  - ADC_CHANNEL_14
  - ADC_CHANNEL_15 (1)
  - ADC_CHANNEL_16 (1)
  - ADC_CHANNEL_17 (3)
  - ADC_CHANNEL_18 (3)
  - ADC_CHANNEL_19 (3)
  - ADC_CHANNEL_20 (3)
  - ADC_CHANNEL_21 (3)
  - ADC_CHANNEL_22 (3)
  - ADC_CHANNEL_VREFINT (2)
  - ADC_CHANNEL_TEMPSENSOR (2)
  - ADC_CHANNEL_VDDA (2)
  - ADC_CHANNEL_VSSA (2)
- __GROUP__: This parameter can be one of the following values:
  - LL_ADC_GROUP_REGULAR

Return value:

- Returned: value can be one of the following values:
  - LL_ADC_AWD_DISABLE
  - LL_ADC_AWD_ALL_CHANNELS_REG
  - LL_ADC_AWD_CHANNEL_0_REG
  - LL_ADC_AWD_CHANNEL_1_REG
  - LL_ADC_AWD_CHANNEL_2_REG
  - LL_ADC_AWD_CHANNEL_3_REG
  - LL_ADC_AWD_CHANNEL_4_REG
  - LL_ADC_AWD_CHANNEL_5_REG
  - LL_ADC_AWD_CHANNEL_6_REG
  - LL_ADC_AWD_CHANNEL_7_REG
  - LL_ADC_AWD_CHANNEL_8_REG
  - LL_ADC_AWD_CHANNEL_9_REG
  - LL_ADC_AWD_CHANNEL_10_REG
  - LL_ADC_AWD_CHANNEL_11_REG
  - LL_ADC_AWD_CHANNEL_12_REG
  - LL_ADC_AWD_CHANNEL_13_REG
  - LL_ADC_AWD_CHANNEL_14_REG
  - LL_ADC_AWD_CHANNEL_15_REG
  - LL_ADC_AWD_CHANNEL_16_REG
  - LL_ADC_AWD_CHANNEL_17_REG
  - LL_ADC_AWD_CHANNEL_18_REG (1)
  - LL_ADC_AWD_CHANNEL_19_REG (1)
  - LL_ADC_AWD_CHANNEL_20_REG (1)
  - LL_ADC_AWD_CHANNEL_21_REG (1)
  - LL_ADC_AWD_CHANNEL_22_REG (1)
  - LL_ADC_AWD_CH_VREFINT_REG
  - LL_ADC_AWD_CH_TEMPSENSOR_REG
  - LL_ADC_AWD_CH_VDDA_REG
  - LL_ADC_AWD_CH_VSSA_REG (1) On STM32C0, parameter not available on all devices: only on STM32C031xx.

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 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

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

Notes:

- To be used with function LL_ADC_ConfigAnalogWDThresholds() or 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 12 bits.

Parameters:

- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B
- __AWD_THRESHOLD_12_BITS__: Value between Min_Data=0x000 and Max_Data=0xFFF

Return value:

- 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_6_bits > = __LL_ADC_ANALOGWD_GET_THRESHOLD_RESOLUTION (LL_ADC_RESOLUTION_8B, LL_ADC_GetAnalogWDThresholds(<ADCx param>, LL_ADC_AWD_THRESHOLD_HIGH));

__LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW

Description:

- Helper macro to get the ADC analog watchdog threshold high or low from raw value containing both thresholds concatenated.

Parameters:

- __AWD_THRESHOLD_TYPE__: This parameter can be one of the following values:
  - LL_ADC_AWD_THRESHOLD_HIGH
  - LL_ADC_AWD_THRESHOLD_LOW
- __AWD_THRESHOLDS__: Value between Min_Data=0x00000000 and Max_Data=0xFFFFFFFF

Return value:

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

Notes:

- To be used with function LL_ADC_GetAnalogWDThresholds(). Example, to get analog watchdog threshold high from the register raw value: __LL_ADC_ANALOGWD_THRESHOLDS_HIGH_LOW(LL_ADC_AWD_THRESHOLD_HIGH, <raw_value_with_both_thresholds>);

__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_RESOLUTION__: This parameter can be one of the following values:
  - 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:

- __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_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_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 12 bits) (unit: digital value).
- __ADC_RESOLUTION__: This parameter can be one of the following values:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

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_VREFANALOG_VOLTAGE

Description:

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

Parameters:

- __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:
  - LL_ADC_RESOLUTION_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

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.

__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:

- __TEMPSENSOR_TYP_AVGSLOPE__: Device datasheet data: Temperature sensor slope typical value (unit: uV/DegCelsius). On this STM32 series, 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 this STM32 series, refer to 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+) value (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_12B
  - LL_ADC_RESOLUTION_10B
  - LL_ADC_RESOLUTION_8B
  - LL_ADC_RESOLUTION_6B

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. : On this STM32 series, calibtation parameter TS_CAL1 can be used to improve calculation accuracy. Refer to TEMPSENSOR_CAL1_ADDR. 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 12 bits (full scale digital value 4095). If not the case, the data must be preliminarily rescaled to an equivalent resolution of 12 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

46 LL BUS Generic Driver

46.1 BUS Firmware driver API description

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

46.1.1 Detailed description of functions

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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBENR DMA1EN LL_AHB1_GRP1_EnableClock
- AHBENR FLASHEN LL_AHB1_GRP1_EnableClock
- AHBENR CRCEN 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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- AHBENR DMA1EN LL_AHB1_GRP1_IsEnabledClock
- AHBENR FLASHEN LL_AHB1_GRP1_IsEnabledClock
- AHBENR CRCEN LL_AHB1_GRP1_IsEnabledClock

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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBENR DMA1EN LL_AHB1_GRP1_DisableClock
- AHBENR FLASHEN LL_AHB1_GRP1_DisableClock
- AHBENR CRCEN 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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBRSTR DMA1RST LL_AHB1_GRP1_ForceReset
- AHBRSTR FLASHRST LL_AHB1_GRP1_ForceReset
- AHBRSTR CRCRST 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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBRSTR DMA1RST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR FLASHRST LL_AHB1_GRP1_ReleaseReset
- AHBRSTR CRCRST 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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_SRAM
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBSMENR DMA1SMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHBSMENR FLASHSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHBSMENR SRAMSMEN LL_AHB1_GRP1_EnableClockStopSleep
- AHBSMENR CRCSMEN LL_AHB1_GRP1_EnableClockStopSleep

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_DMA1
  - LL_AHB1_GRP1_PERIPH_FLASH
  - LL_AHB1_GRP1_PERIPH_SRAM
  - LL_AHB1_GRP1_PERIPH_CRC

Return values

- **None:**

Reference Manual to LL API cross reference:

- AHBSMENR DMA1SMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHBSMENR FLASHSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHBSMENR SRAMSMEN LL_AHB1_GRP1_DisableClockStopSleep
- AHBSMENR CRCSMEN LL_AHB1_GRP1_DisableClockStopSleep

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_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBENR1 TIM3EN LL_APB1_GRP1_EnableClock
- APBENR1 RTCAPBEN LL_APB1_GRP1_EnableClock
- APBENR1 WWDGEN LL_APB1_GRP1_EnableClock
- APBENR1 I2C1EN LL_APB1_GRP1_EnableClock
- APBENR1 DBGEN LL_APB1_GRP1_EnableClock
- APBENR1 PWREN LL_APB1_GRP1_EnableClock
- APBENR1 TIM2EN LL_APB1_GRP1_EnableClock
- (*) APBENR1 SPI2EN LL_APB1_GRP1_EnableClock
- (*) APBENR1 I2C2EN LL_APB1_GRP1_EnableClock
- (*) APBENR1 USBEN LL_APB1_GRP1_EnableClock
- (*) APBENR1 FDCAN1EN LL_APB1_GRP1_EnableClock
- (*) APBENR1 USART3EN LL_APB1_GRP1_EnableClock
- (*) APBENR1 USART4EN LL_APB1_GRP1_EnableClock (*)

LL_APB1_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 GRP1 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **State:** of Periphs (1 or 0).

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBENR1 TIM3EN LL_APB1_GRP1_IsEnabledClock
- APBENR1 RTCAPBEN LL_APB1_GRP1_IsEnabledClock
- APBENR1 WWDGEN LL_APB1_GRP1_IsEnabledClock
- APBENR1 USART2EN LL_APB1_GRP1_IsEnabledClock
- APBENR1 I2C1EN LL_APB1_GRP1_IsEnabledClock
- APBENR1 DBGEN LL_APB1_GRP1_IsEnabledClock
- APBENR1 PWREN LL_APB1_GRP1_IsEnabledClock
- APBENR1 TIM2EN LL_APB1_GRP1_IsEnabledClock
- (*) APBENR1 SPI2EN LL_APB1_GRP1_IsEnabledClock
- (*) APBENR1 I2C2EN LL_APB1_GRP1_IsEnabledClock
- (*) APBENR1 USBEN LL_APB1_GRP1_IsEnabledClock
- (*) APBENR1 FDCAN1EN LL_APB1_GRP1_IsEnabledClock
- (*) APBENR1 USART3EN LL_APB1_GRP1_IsEnabledClock
- (*) APBENR1 USART4EN LL_APB1_GRP1_IsEnabledClock (*)

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 GRP1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_USART2
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBENR1 TIM3EN LL_APB1_GRP1_DisableClock
- APBENR1 RTCAPBEN LL_APB1_GRP1_DisableClock
- APBENR1 WWDGEN LL_APB1_GRP1_DisableClock
- APBENR1 USART2EN LL_APB1_GRP1_DisableClock
- APBENR1 I2C1EN LL_APB1_GRP1_DisableClock
- APBENR1 DBGEN LL_APB1_GRP1_DisableClock
- APBENR1 PWREN LL_APB1_GRP1_DisableClock
- APBENR1 TIM2EN LL_APB1_GRP1_DisableClock
- (*) APBENR1 SPI2EN LL_APB1_GRP1_DisableClock
- (*) APBENR1 I2C2EN LL_APB1_GRP1_DisableClock
- (*) APBENR1 USBEN LL_APB1_GRP1_DisableClock
- (*) APBENR1 FDACN1EN LL_APB1_GRP1_DisableClock
- (*) APBENR1 USART3EN LL_APB1_GRP1_DisableClock
- (*) APBENR1 USART4EN LL_APB1_GRP1_DisableClock (*)

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 GRP1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBRSTR1 TIM3RST LL_APB1_GRP1_ForceReset
- APBRSTR1 RTCRST LL_APB1_GRP1_ForceReset
- APBRSTR1 WWDGRST LL_APB1_GRP1_ForceReset
- APBRSTR1 I2C1RST LL_APB1_GRP1_ForceReset
- APBRSTR1 DBGRST LL_APB1_GRP1_ForceReset
- APBRSTR1 PWRRST LL_APB1_GRP1_ForceReset
- APBRSTR1 TIM2RST LL_APB1_GRP1_ForceReset
- (*) APBRSTR1 SPI2RST LL_APB1_GRP1_ForceReset
- (*) APBRSTR1 I2C2RST LL_APB1_GRP1_ForceReset
- (*) APBRSTR1 USBRST LL_APB1_GRP1_ForceReset
- (*) APBRSTR1 FDCAN1RST LL_APB1_GRP1_ForceReset
- (*) APBRSTR1 USART3RST LL_APB1_GRP1_ForceReset
- (*) APBRSTR1 USART4RST LL_APB1_GRP1_ForceReset (*)

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 GRP1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBRSTR1 TIM3RST LL_APB1_GRP1_ReleaseReset
- APBRSTR1 RTCRST LL_APB1_GRP1_ReleaseReset
- APBRSTR1 WWDGRST LL_APB1_GRP1_ReleaseReset
- APBRSTR1 I2C1RST LL_APB1_GRP1_ReleaseReset
- APBRSTR1 DBGRST LL_APB1_GRP1_ReleaseReset
- APBRSTR1 PWRRST LL_APB1_GRP1_ReleaseReset
- APBRSTR1 TIM2RST LL_APB1_GRP1_ReleaseReset
- (*) APBRSTR1 SPI2RST LL_APB1_GRP1_ReleaseReset
- (*) APBRSTR1 I2C2RST LL_APB1_GRP1_ReleaseReset
- (*) APBRSTR1 USBRST LL_APB1_GRP1_ReleaseReset
- (*) APBRSTR1 FDCAN1RST LL_APB1_GRP1_ReleaseReset
- (*) APBRSTR1 USART3RST LL_APB1_GRP1_ReleaseReset
- (*) APBRSTR1 USART4RST LL_APB1_GRP1_ReleaseReset (*)

LL_APB1_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB1 GRP1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBSMENR1 TIM3SMEN LL_APB1_GRP1_EnableClockStopSleep
- APBSMENR1 RTCAPBSMEN LL_APB1_GRP1_EnableClockStopSleep
- APBSMENR1 WWDGSMEN LL_APB1_GRP1_EnableClockStopSleep
- APBSMENR1 I2C1SMEN LL_APB1_GRP1_EnableClockStopSleep
- APBSMENR1 DBGSMEN LL_APB1_GRP1_EnableClockStopSleep
- APBSMENR1 PWRSMEN LL_APB1_GRP1_EnableClockStopSleep
- APBSMENR1 TIM2SMEN LL_APB1_GRP1_EnableClockStopSleep
- (*) APBSMENR1 SPI2SMEN LL_APB1_GRP1_EnableClockStopSleep
- (*) APBSMENR1 I2C2SMEN LL_APB1_GRP1_EnableClockStopSleep
- (*) APBSMENR1 USBSMEN LL_APB1_GRP1_EnableClockStopSleep
- (*) APBSMENR1 FDCAN1SMEN LL_APB1_GRP1_EnableClockStopSleep
- (*) APBSMENR1 USART3SMEN LL_APB1_GRP1_EnableClockStopSleep
- (*) APBSMENR1 USART4SMEN LL_APB1_GRP1_EnableClockStopSleep (*)

LL_APB1_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB1 GRP1 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_TIM3
  - LL_APB1_GRP1_PERIPH_RTC
  - LL_APB1_GRP1_PERIPH_WWDG
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_DBGMCU
  - LL_APB1_GRP1_PERIPH_PWR
  - LL_APB1_GRP1_PERIPH_TIM2 (*)
  - LL_APB1_GRP1_PERIPH_SPI2 (*)
  - LL_APB1_GRP1_PERIPH_I2C2 (*)
  - LL_APB1_GRP1_PERIPH_USB (*)
  - LL_APB1_GRP1_PERIPH_FDCAN1 (*)
  - LL_APB1_GRP1_PERIPH_USART3 (*)
  - LL_APB1_GRP1_PERIPH_USART4 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBSMENR1 TIM3SMEN LL_APB1_GRP1_DisableClockStopSleep
- APBSMENR1 RTCAPBSMEN LL_APB1_GRP1_DisableClockStopSleep
- APBSMENR1 WWDGSMEN LL_APB1_GRP1_DisableClockStopSleep
- APBSMENR1 I2C1SMEN LL_APB1_GRP1_DisableClockStopSleep
- APBSMENR1 DBGSMEN LL_APB1_GRP1_DisableClockStopSleep
- APBSMENR1 PWRSMEN LL_APB1_GRP1_DisableClockStopSleep
- APBSMENR1 TIM2SMEN LL_APB1_GRP1_DisableClockStopSleep
- (*) APBSMENR1 SPI2SMEN LL_APB1_GRP1_DisableClockStopSleep
- (*) APBSMENR1 I2C2SMEN LL_APB1_GRP1_DisableClockStopSleep
- (*) APBSMENR1 USBSMEN LL_APB1_GRP1_DisableClockStopSleep
- (*) APBSMENR1 FDCAN1SMEN LL_APB1_GRP1_DisableClockStopSleep
- (*) APBSMENR1 USART3SMEN LL_APB1_GRP1_DisableClockStopSleep
- (*) APBSMENR1 USART4SMEN LL_APB1_GRP1_DisableClockStopSleep (*)

LL_APB1_GRP2_EnableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClock (uint32_t Periphs)

Function description

Enable APB1 GRP2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBENR2 SYSCFGEN LL_APB1_GRP2_EnableClock
- APBENR2 TIM1EN LL_APB1_GRP2_EnableClock
- APBENR2 SPI1EN LL_APB1_GRP2_EnableClock
- APBENR2 USART1EN LL_APB1_GRP2_EnableClock
- APBENR2 TIM14EN LL_APB1_GRP2_EnableClock
- APBENR2 TIM16EN LL_APB1_GRP2_EnableClock
- APBENR2 TIM17EN LL_APB1_GRP2_EnableClock
- APBENR2 ADCEN LL_APB1_GRP2_EnableClock
- APBENR2 TIM15EN LL_APB1_GRP2_EnableClock (*)

LL_APB1_GRP2_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB1_GRP2_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB1 GRP2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **State:** of Periphs (1 or 0).

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBENR2 SYSCFGEN LL_APB1_GRP2_IsEnabledClock
- APBENR2 TIM1EN LL_APB1_GRP2_IsEnabledClock
- APBENR2 SPI1EN LL_APB1_GRP2_IsEnabledClock
- APBENR2 USART1EN LL_APB1_GRP2_IsEnabledClock
- APBENR2 TIM14EN LL_APB1_GRP2_IsEnabledClock
- APBENR2 TIM16EN LL_APB1_GRP2_IsEnabledClock
- APBENR2 TIM17EN LL_APB1_GRP2_IsEnabledClock
- APBENR2 ADCEN LL_APB1_GRP2_IsEnabledClock
- APBENR2 TIM15EN LL_APB1_GRP2_IsEnabledClock (*)

LL_APB1_GRP2_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClock (uint32_t Periphs)

Function description

Disable APB1 GRP2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBENR2 SYSCFGEN LL_APB1_GRP2_DisableClock
- APBENR2 TIM1EN LL_APB1_GRP2_DisableClock
- APBENR2 SPI1EN LL_APB1_GRP2_DisableClock
- APBENR2 USART1EN LL_APB1_GRP2_DisableClock
- APBENR2 TIM14EN LL_APB1_GRP2_DisableClock
- APBENR2 TIM16EN LL_APB1_GRP2_DisableClock
- APBENR2 TIM17EN LL_APB1_GRP2_DisableClock
- APBENR2 ADCEN LL_APB1_GRP2_DisableClock
- APBENR2 TIM15EN LL_APB1_GRP2_DisableClock (*)

LL_APB1_GRP2_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ForceReset (uint32_t Periphs)

Function description

Force APB1 GRP2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBRSTR2 SYSCFGRST LL_APB1_GRP2_ForceReset
- APBRSTR2 TIM1RST LL_APB1_GRP2_ForceReset
- APBRSTR2 SPI1RST LL_APB1_GRP2_ForceReset
- APBRSTR2 USART1RST LL_APB1_GRP2_ForceReset
- APBRSTR2 TIM14RST LL_APB1_GRP2_ForceReset
- APBRSTR2 TIM16RST LL_APB1_GRP2_ForceReset
- APBRSTR2 TIM17RST LL_APB1_GRP2_ForceReset
- APBRSTR2 ADCRST LL_APB1_GRP2_ForceReset
- APBRSTR2 TIM15RST LL_APB1_GRP2_ForceReset (*)

LL_APB1_GRP2_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP2_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 GRP2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_ALL
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBRSTR2 SYSCFGRST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 TIM1RST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 SPI1RST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 USART1RST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 TIM14RST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 TIM16RST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 TIM17RST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 ADCRST LL_APB1_GRP2_ReleaseReset
- APBRSTR2 TIM15RST LL_APB1_GRP2_ReleaseReset (*)

LL_APB1_GRP2_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable APB1 GRP2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBSMENR2 SYSCFGSMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 TIM1SMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 SPI1SMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 USART1SMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 TIM14SMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 TIM16SMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 TIM17SMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 ADCSMEN LL_APB1_GRP2_EnableClockStopSleep
- APBSMENR2 TIM15SMEN LL_APB1_GRP2_EnableClockStopSleep (*)

LL_APB1_GRP2_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_APB1_GRP2_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable APB1 GRP2 peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP2_PERIPH_SYSCFG
  - LL_APB1_GRP2_PERIPH_TIM1
  - LL_APB1_GRP2_PERIPH_SPI1
  - LL_APB1_GRP2_PERIPH_USART1
  - LL_APB1_GRP2_PERIPH_TIM14
  - LL_APB1_GRP2_PERIPH_TIM16
  - LL_APB1_GRP2_PERIPH_TIM17
  - LL_APB1_GRP2_PERIPH_ADC
  - LL_APB1_GRP2_PERIPH_TIM15 (*)

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- APBSMENR2 SYSCFGSMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 TIM1SMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 SPI1SMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 USART1SMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 TIM14SMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 TIM16SMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 TIM17SMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 ADCSMEN LL_APB1_GRP2_DisableClockStopSleep
- APBSMENR2 TIM15SMEN LL_APB1_GRP2_DisableClockStopSleep (*)

LL_IOP_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_IOP_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable IOP peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **None:**

Reference Manual to LL API cross reference:

- IOPENR GPIOAEN LL_IOP_GRP1_EnableClock
- IOPENR GPIOBEN LL_IOP_GRP1_EnableClock
- IOPENR GPIOCEN LL_IOP_GRP1_EnableClock
- IOPENR GPIODEN LL_IOP_GRP1_EnableClock
- IOPENR GPIOFEN LL_IOP_GRP1_EnableClock

LL_IOP_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_IOP_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if IOP peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **State:** of Periphs (1 or 0).

Reference Manual to LL API cross reference:

- IOPENR GPIOAEN LL_IOP_GRP1_IsEnabledClock
- IOPENR GPIOBEN LL_IOP_GRP1_IsEnabledClock
- IOPENR GPIOCEN LL_IOP_GRP1_IsEnabledClock
- IOPENR GPIODEN LL_IOP_GRP1_IsEnabledClock
- IOPENR GPIOFEN LL_IOP_GRP1_IsEnabledClock

LL_IOP_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_IOP_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable IOP peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **None:**

Reference Manual to LL API cross reference:

- IOPENR GPIOAEN LL_IOP_GRP1_DisableClock
- IOPENR GPIOBEN LL_IOP_GRP1_DisableClock
- IOPENR GPIOCEN LL_IOP_GRP1_DisableClock
- IOPENR GPIODEN LL_IOP_GRP1_DisableClock
- IOPENR GPIOFEN LL_IOP_GRP1_DisableClock

LL_IOP_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_IOP_GRP1_ForceReset (uint32_t Periphs)

Function description

Disable IOP peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_ALL
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **None:**

Reference Manual to LL API cross reference:

- IOPRSTR GPIOARST LL_IOP_GRP1_ForceReset
- IOPRSTR GPIOBRST LL_IOP_GRP1_ForceReset
- IOPRSTR GPIOCRST LL_IOP_GRP1_ForceReset
- IOPRSTR GPIODRST LL_IOP_GRP1_ForceReset
- IOPRSTR GPIOFRST LL_IOP_GRP1_ForceReset

LL_IOP_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_IOP_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release IOP peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_ALL
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **None:**

Reference Manual to LL API cross reference:

- IOPRSTR GPIOARST LL_IOP_GRP1_ReleaseReset
- IOPRSTR GPIOBRST LL_IOP_GRP1_ReleaseReset
- IOPRSTR GPIOCRST LL_IOP_GRP1_ReleaseReset
- IOPRSTR GPIODRST LL_IOP_GRP1_ReleaseReset
- IOPRSTR GPIOFRST LL_IOP_GRP1_ReleaseReset

LL_IOP_GRP1_EnableClockStopSleep

Function name

__STATIC_INLINE void LL_IOP_GRP1_EnableClockStopSleep (uint32_t Periphs)

Function description

Enable IOP peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **None:**

Reference Manual to LL API cross reference:

- IOPSMENR GPIOASMEN LL_IOP_GRP1_EnableClockStopSleep
- IOPSMENR GPIOBSMEN LL_IOP_GRP1_EnableClockStopSleep
- IOPSMENR GPIOCSMEN LL_IOP_GRP1_EnableClockStopSleep
- IOPSMENR GPIODSMEN LL_IOP_GRP1_EnableClockStopSleep
- IOPSMENR GPIOFSMEN LL_IOP_GRP1_EnableClockStopSleep

LL_IOP_GRP1_DisableClockStopSleep

Function name

__STATIC_INLINE void LL_IOP_GRP1_DisableClockStopSleep (uint32_t Periphs)

Function description

Disable IOP peripheral clocks in Sleep and Stop modes.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_IOP_GRP1_PERIPH_GPIOA
  - LL_IOP_GRP1_PERIPH_GPIOB
  - LL_IOP_GRP1_PERIPH_GPIOC
  - LL_IOP_GRP1_PERIPH_GPIOD
  - LL_IOP_GRP1_PERIPH_GPIOF

Return values

- **None:**

Reference Manual to LL API cross reference:

- IOPSMENR GPIOASMEN LL_IOP_GRP1_DisableClockStopSleep
- IOPSMENR GPIOBSMEN LL_IOP_GRP1_DisableClockStopSleep
- IOPSMENR GPIOCSMEN LL_IOP_GRP1_DisableClockStopSleep
- IOPSMENR GPIODSMEN LL_IOP_GRP1_DisableClockStopSleep
- IOPSMENR GPIOFSMEN LL_IOP_GRP1_DisableClockStopSleep

46.2 BUS Firmware driver defines

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

46.2.1 BUS

BUS

AHB1 GRP1 PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_DMA1

LL_AHB1_GRP1_PERIPH_FLASH

LL_AHB1_GRP1_PERIPH_SRAM

LL_AHB1_GRP1_PERIPH_CRC

APB1 GRP1 PERIPH

LL_APB1_GRP1_PERIPH_ALL

LL_APB1_GRP1_PERIPH_TIM2

LL_APB1_GRP1_PERIPH_TIM3

LL_APB1_GRP1_PERIPH_RTC

LL_APB1_GRP1_PERIPH_WWDG

LL_APB1_GRP1_PERIPH_FDCAN1

LL_APB1_GRP1_PERIPH_SPI2

LL_APB1_GRP1_PERIPH_USART2

LL_APB1_GRP1_PERIPH_USART3

LL_APB1_GRP1_PERIPH_USART4

LL_APB1_GRP1_PERIPH_I2C1

LL_APB1_GRP1_PERIPH_I2C2

LL_APB1_GRP1_PERIPH_DBGMCU

LL_APB1_GRP1_PERIPH_PWR

APB1 GRP2 PERIPH

LL_APB1_GRP2_PERIPH_ALL

LL_APB1_GRP2_PERIPH_SYSCFG

LL_APB1_GRP2_PERIPH_TIM1

LL_APB1_GRP2_PERIPH_SPI1

LL_APB1_GRP2_PERIPH_USART1

LL_APB1_GRP2_PERIPH_TIM14

LL_APB1_GRP2_PERIPH_TIM15

LL_APB1_GRP2_PERIPH_TIM16

LL_APB1_GRP2_PERIPH_TIM17

LL_APB1_GRP2_PERIPH_ADC

LL_APB2_GRP2_PERIPH_ALL

LL_APB2_GRP1_PERIPH_SYSCFG

LL_APB2_GRP1_PERIPH_TIM1

LL_APB2_GRP1_PERIPH_SPI1

LL_APB2_GRP1_PERIPH_USART1

LL_APB2_GRP1_PERIPH_TIM14

LL_APB2_GRP1_PERIPH_TIM16

LL_APB2_GRP1_PERIPH_TIM17

LL_APB2_GRP1_PERIPH_ADC

IOP GRP1 PERIPH

LL_IOP_GRP1_PERIPH_ALL

LL_IOP_GRP1_PERIPH_GPIOA

LL_IOP_GRP1_PERIPH_GPIOB

LL_IOP_GRP1_PERIPH_GPIOC

LL_IOP_GRP1_PERIPH_GPIOD

LL_IOP_GRP1_PERIPH_GPIOF

APB1_GRP2

LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClockStopSleep

LL_APB2_GRP1_DisableClockStopSleep

47 LL CORTEX Generic Driver

47.1 CORTEX Firmware driver API description

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

47.1.1 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_HCLK_DIV8
  - 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_HCLK_DIV8
  - 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_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 number.

Return values

- **Value:** should be equal to 0xC for Cortex-M0+ devices

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 0xC60 for Cortex-M0+

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 Options)

Function description

Enable MPU with input options.

Parameters

- **Options:** 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_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_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:**

Reference Manual to LL API cross reference:

- MPU_RASR ENABLE LL_MPU_EnableRegion

LL_MPU_ConfigRegion

Function name

__STATIC_INLINE void LL_MPU_ConfigRegion (uint32_t Region, uint32_t SubRegionDisable, uint32_t Address, uint32_t Attributes)

Function description

Configure 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
- **Address:** Value of region base address
- **SubRegionDisable:** Sub-region disable value between Min_Data = 0x00 and Max_Data = 0xFF
- **Attributes:** This parameter can be a combination of the following values:
  - LL_MPU_REGION_SIZE_256B or LL_MPU_REGION_SIZE_512B or LL_MPU_REGION_SIZE_1KB or LL_MPU_REGION_SIZE_2KB or LL_MPU_REGION_SIZE_4KB or LL_MPU_REGION_SIZE_8KB or LL_MPU_REGION_SIZE_16KB or LL_MPU_REGION_SIZE_32KB
  - LL_MPU_REGION_NO_ACCESS or LL_MPU_REGION_PRIV_RW or LL_MPU_REGION_PRIV_RW_URO or LL_MPU_REGION_FULL_ACCESS or LL_MPU_REGION_PRIV_RO or LL_MPU_REGION_PRIV_RO_URO
  - LL_MPU_TEX_LEVEL0 or LL_MPU_TEX_LEVEL1 or LL_MPU_TEX_LEVEL2 or LL_MPU_TEX_LEVEL4
  - LL_MPU_INSTRUCTION_ACCESS_ENABLE or LL_MPU_INSTRUCTION_ACCESS_DISABLE
  - LL_MPU_ACCESS_SHAREABLE or LL_MPU_ACCESS_NOT_SHAREABLE
  - LL_MPU_ACCESS_CACHEABLE or LL_MPU_ACCESS_NOT_CACHEABLE
  - LL_MPU_ACCESS_BUFFERABLE or LL_MPU_ACCESS_NOT_BUFFERABLE

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_ConfigRegion
- MPU_RBAR REGION LL_MPU_ConfigRegion
- MPU_RBAR ADDR LL_MPU_ConfigRegion
- MPU_RASR XN LL_MPU_ConfigRegion
- MPU_RASR AP LL_MPU_ConfigRegion
- MPU_RASR S LL_MPU_ConfigRegion
- MPU_RASR C LL_MPU_ConfigRegion
- MPU_RASR B LL_MPU_ConfigRegion
- MPU_RASR SIZE LL_MPU_ConfigRegion

LL_MPU_DisableRegion

Function name

__STATIC_INLINE void LL_MPU_DisableRegion (uint32_t Region)

Function description

Disable 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

Return values

- **None:**

Reference Manual to LL API cross reference:

- MPU_RNR REGION LL_MPU_DisableRegion
- MPU_RASR ENABLE LL_MPU_DisableRegion

47.2 CORTEX Firmware driver defines

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

47.2.1 CORTEX

CORTEX

MPU Bufferable Access

LL_MPU_ACCESS_BUFFERABLE

Bufferable memory attribute

LL_MPU_ACCESS_NOT_BUFFERABLE

Not Bufferable memory attribute

MPU Cacheable Access

LL_MPU_ACCESS_CACHEABLE

Cacheable memory attribute

LL_MPU_ACCESS_NOT_CACHEABLE

Not Cacheable memory attribute

SYSTICK Clock Source

LL_SYSTICK_CLKSOURCE_HCLK_DIV8

AHB clock divided by 8 selected as SysTick clock source.

LL_SYSTICK_CLKSOURCE_HCLK

AHB clock selected as SysTick clock source.

MPU Control

LL_MPU_CTRL_HFNMI_PRIVDEF_NONE

Disable NMI and privileged SW access

LL_MPU_CTRL_HARDFAULT_NMI

Enables the operation of MPU during hard fault, NMI, and FAULTMASK handlers

LL_MPU_CTRL_PRIVILEGED_DEFAULT

Enable privileged software access to default memory map

LL_MPU_CTRL_HFNMI_PRIVDEF

Enable NMI and privileged SW access

MPU Instruction Access

LL_MPU_INSTRUCTION_ACCESS_ENABLE

Instruction fetches enabled

LL_MPU_INSTRUCTION_ACCESS_DISABLE

Instruction fetches disabled

MPU Region Number

LL_MPU_REGION_NUMBER0

REGION Number 0

LL_MPU_REGION_NUMBER1

REGION Number 1

LL_MPU_REGION_NUMBER2

REGION Number 2

LL_MPU_REGION_NUMBER3

REGION Number 3

LL_MPU_REGION_NUMBER4

REGION Number 4

LL_MPU_REGION_NUMBER5

REGION Number 5

LL_MPU_REGION_NUMBER6

REGION Number 6

LL_MPU_REGION_NUMBER7

REGION Number 7

MPU Region Privileges

LL_MPU_REGION_NO_ACCESS

No access

LL_MPU_REGION_PRIV_RW

RW privileged (privileged access only)

LL_MPU_REGION_PRIV_RW_URO

RW privileged - RO user (Write in a user program generates a fault)

LL_MPU_REGION_FULL_ACCESS

RW privileged & user (Full access)

LL_MPU_REGION_PRIV_RO

RO privileged (privileged read only)

LL_MPU_REGION_PRIV_RO_URO

RO privileged & user (read only)

MPU Region Size

LL_MPU_REGION_SIZE_256B

256B Size of the MPU protection region

LL_MPU_REGION_SIZE_512B

512B Size of the MPU protection region

LL_MPU_REGION_SIZE_1KB

1KB Size of the MPU protection region

LL_MPU_REGION_SIZE_2KB

2KB Size of the MPU protection region

LL_MPU_REGION_SIZE_4KB

4KB Size of the MPU protection region

LL_MPU_REGION_SIZE_8KB

8KB Size of the MPU protection region

LL_MPU_REGION_SIZE_16KB

16KB Size of the MPU protection region

LL_MPU_REGION_SIZE_32KB

32KB Size of the MPU protection region

LL_MPU_REGION_SIZE_64KB

64KB Size of the MPU protection region

LL_MPU_REGION_SIZE_128KB

128KB Size of the MPU protection region

LL_MPU_REGION_SIZE_256KB

256KB Size of the MPU protection region

LL_MPU_REGION_SIZE_512KB

512KB Size of the MPU protection region

LL_MPU_REGION_SIZE_1MB

1MB Size of the MPU protection region

LL_MPU_REGION_SIZE_2MB

2MB Size of the MPU protection region

LL_MPU_REGION_SIZE_4MB

4MB Size of the MPU protection region

LL_MPU_REGION_SIZE_8MB

8MB Size of the MPU protection region

LL_MPU_REGION_SIZE_16MB

16MB Size of the MPU protection region

LL_MPU_REGION_SIZE_32MB

32MB Size of the MPU protection region

LL_MPU_REGION_SIZE_64MB

64MB Size of the MPU protection region

LL_MPU_REGION_SIZE_128MB

128MB Size of the MPU protection region

LL_MPU_REGION_SIZE_256MB

256MB Size of the MPU protection region

LL_MPU_REGION_SIZE_512MB

512MB Size of the MPU protection region

LL_MPU_REGION_SIZE_1GB

1GB Size of the MPU protection region

LL_MPU_REGION_SIZE_2GB

2GB Size of the MPU protection region

LL_MPU_REGION_SIZE_4GB

4GB Size of the MPU protection region

MPU Shareable Access

LL_MPU_ACCESS_SHAREABLE

Shareable memory attribute

LL_MPU_ACCESS_NOT_SHAREABLE

Not Shareable memory attribute

MPU TEX Level

LL_MPU_TEX_LEVEL0

b000 for TEX bits

LL_MPU_TEX_LEVEL1

b001 for TEX bits

LL_MPU_TEX_LEVEL2

b010 for TEX bits

LL_MPU_TEX_LEVEL4

b100 for TEX bits

48 LL CRC Generic Driver

48.1 CRC Firmware driver API description

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

48.1.1 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

48.2 CRC Firmware driver defines

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

48.2.1 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

49 LL DMAMUX Generic Driver

49.1 DMAMUX Firmware driver API description

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

49.1.1 Detailed description of functions

LL_DMAMUX_SetRequestID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestID (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Request)

Function description

Set DMAMUX request ID for DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX (*)
  - LL_DMAMUX_REQ_I2C2_TX (*)
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX (*)
  - LL_DMAMUX_REQ_SPI2_TX (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_TRIG_COM
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM2_CH1 (*)
  - LL_DMAMUX_REQ_TIM2_CH2 (*)
  - LL_DMAMUX_REQ_TIM2_CH3 (*)
  - LL_DMAMUX_REQ_TIM2_CH4 (*)
  - LL_DMAMUX_REQ_TIM2_TRIG_COM (*)
  - LL_DMAMUX_REQ_TIM2_UP (*)
  - LL_DMAMUX_REQ_TIM3_CH1
  - LL_DMAMUX_REQ_TIM3_CH2
  - LL_DMAMUX_REQ_TIM3_CH3
  - LL_DMAMUX_REQ_TIM3_CH4
  - LL_DMAMUX_REQ_TIM3_TRIG
  - LL_DMAMUX_REQ_TIM3_UP
  - LL_DMAMUX_REQ_TIM16_CH1
  - LL_DMAMUX_REQ_TIM16_TRIG_COM
  - LL_DMAMUX_REQ_TIM16_UP
  - LL_DMAMUX_REQ_TIM17_CH1
  - LL_DMAMUX_REQ_TIM17_TRIG_COM
  - LL_DMAMUX_REQ_TIM17_UP
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX (*)
  - LL_DMAMUX_REQ_USART3_TX (*)
  - LL_DMAMUX_REQ_USART4_RX (*)
  - LL_DMAMUX_REQ_USART4_TX (*)

Return values

- **None:**

Notes

- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_SetRequestID

LL_DMAMUX_GetRequestID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestID (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get DMAMUX request ID for DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX (*)
  - LL_DMAMUX_REQ_I2C2_TX (*)
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX (*)
  - LL_DMAMUX_REQ_SPI2_TX (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_TRIG_COM
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM2_CH1 (*)
  - LL_DMAMUX_REQ_TIM2_CH2 (*)
  - LL_DMAMUX_REQ_TIM2_CH3 (*)
  - LL_DMAMUX_REQ_TIM2_CH4 (*)
  - LL_DMAMUX_REQ_TIM2_TRIG_COM (*)
  - LL_DMAMUX_REQ_TIM2_UP (*)
  - LL_DMAMUX_REQ_TIM3_CH1
  - LL_DMAMUX_REQ_TIM3_CH2
  - LL_DMAMUX_REQ_TIM3_CH3
  - LL_DMAMUX_REQ_TIM3_CH4
  - LL_DMAMUX_REQ_TIM3_TRIG
  - LL_DMAMUX_REQ_TIM3_UP
  - LL_DMAMUX_REQ_TIM16_CH1
  - LL_DMAMUX_REQ_TIM16_TRIG_COM
  - LL_DMAMUX_REQ_TIM16_UP
  - LL_DMAMUX_REQ_TIM17_CH1
  - LL_DMAMUX_REQ_TIM17_TRIG_COM
  - LL_DMAMUX_REQ_TIM17_UP
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX (*)
  - LL_DMAMUX_REQ_USART3_TX (*)
  - LL_DMAMUX_REQ_USART4_RX (*)
  - LL_DMAMUX_REQ_USART4_TX (*)

Notes

- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_GetRequestID

LL_DMAMUX_SetSyncRequestNb

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncRequestNb (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t RequestNb)

Function description

Set the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)
- **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR NBREQ LL_DMAMUX_SetSyncRequestNb

LL_DMAMUX_GetSyncRequestNb

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncRequestNb (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get the number of DMA request that will be autorized after a synchronization event and/or the number of DMA request needed to generate an event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **Between:** Min_Data = 1 and Max_Data = 32

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR NBREQ LL_DMAMUX_GetSyncRequestNb

LL_DMAMUX_SetSyncPolarity

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncPolarity (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t Polarity)

Function description

Set the polarity of the signal on which the DMA request is synchronized.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)
- **Polarity:** This parameter can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SPOL LL_DMAMUX_SetSyncPolarity

LL_DMAMUX_GetSyncPolarity

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncPolarity (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get the polarity of the signal on which the DMA request is synchronized.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_SYNC_NO_EVENT
  - LL_DMAMUX_SYNC_POL_RISING
  - LL_DMAMUX_SYNC_POL_FALLING
  - LL_DMAMUX_SYNC_POL_RISING_FALLING

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SPOL LL_DMAMUX_GetSyncPolarity

LL_DMAMUX_EnableEventGeneration

Function name

__STATIC_INLINE void LL_DMAMUX_EnableEventGeneration (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the Event Generation on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_EnableEventGeneration

LL_DMAMUX_DisableEventGeneration

Function name

__STATIC_INLINE void LL_DMAMUX_DisableEventGeneration (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the Event Generation on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_DisableEventGeneration

LL_DMAMUX_IsEnabledEventGeneration

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledEventGeneration (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the Event Generation on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR EGE LL_DMAMUX_IsEnabledEventGeneration

LL_DMAMUX_EnableSync

Function name

__STATIC_INLINE void LL_DMAMUX_EnableSync (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the synchronization mode.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_EnableSync

LL_DMAMUX_DisableSync

Function name

__STATIC_INLINE void LL_DMAMUX_DisableSync (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the synchronization mode.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_DisableSync

LL_DMAMUX_IsEnabledSync

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledSync (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the synchronization mode is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SE LL_DMAMUX_IsEnabledSync

LL_DMAMUX_SetSyncID

Function name

__STATIC_INLINE void LL_DMAMUX_SetSyncID (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel, uint32_t SyncID)

Function description

Set DMAMUX synchronization ID on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)
- **SyncID:** This parameter can be one of the following values:
  - LL_DMAMUX_SYNC_EXTI_LINE0
  - LL_DMAMUX_SYNC_EXTI_LINE1
  - LL_DMAMUX_SYNC_EXTI_LINE2
  - LL_DMAMUX_SYNC_EXTI_LINE3
  - LL_DMAMUX_SYNC_EXTI_LINE4
  - LL_DMAMUX_SYNC_EXTI_LINE5
  - LL_DMAMUX_SYNC_EXTI_LINE6
  - LL_DMAMUX_SYNC_EXTI_LINE7
  - LL_DMAMUX_SYNC_EXTI_LINE8
  - LL_DMAMUX_SYNC_EXTI_LINE9
  - LL_DMAMUX_SYNC_EXTI_LINE10
  - LL_DMAMUX_SYNC_EXTI_LINE11
  - LL_DMAMUX_SYNC_EXTI_LINE12
  - LL_DMAMUX_SYNC_EXTI_LINE13
  - LL_DMAMUX_SYNC_EXTI_LINE14
  - LL_DMAMUX_SYNC_EXTI_LINE15
  - LL_DMAMUX_SYNC_DMAMUX_CH0
  - LL_DMAMUX_SYNC_DMAMUX_CH1
  - LL_DMAMUX_SYNC_DMAMUX_CH2
  - LL_DMAMUX_SYNC_DMAMUX_CH3 (*)
  - LL_DMAMUX_SYNC_TIM14_OC

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SYNC_ID LL_DMAMUX_SetSyncID

LL_DMAMUX_GetSyncID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetSyncID (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Get DMAMUX synchronization ID on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_SYNC_EXTI_LINE0
  - LL_DMAMUX_SYNC_EXTI_LINE1
  - LL_DMAMUX_SYNC_EXTI_LINE2
  - LL_DMAMUX_SYNC_EXTI_LINE3
  - LL_DMAMUX_SYNC_EXTI_LINE4
  - LL_DMAMUX_SYNC_EXTI_LINE5
  - LL_DMAMUX_SYNC_EXTI_LINE6
  - LL_DMAMUX_SYNC_EXTI_LINE7
  - LL_DMAMUX_SYNC_EXTI_LINE8
  - LL_DMAMUX_SYNC_EXTI_LINE9
  - LL_DMAMUX_SYNC_EXTI_LINE10
  - LL_DMAMUX_SYNC_EXTI_LINE11
  - LL_DMAMUX_SYNC_EXTI_LINE12
  - LL_DMAMUX_SYNC_EXTI_LINE13
  - LL_DMAMUX_SYNC_EXTI_LINE14
  - LL_DMAMUX_SYNC_EXTI_LINE15
  - LL_DMAMUX_SYNC_DMAMUX_CH0
  - LL_DMAMUX_SYNC_DMAMUX_CH1
  - LL_DMAMUX_SYNC_DMAMUX_CH2
  - LL_DMAMUX_SYNC_DMAMUX_CH3 (*)
  - LL_DMAMUX_SYNC_TIM14_OC

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SYNC_ID LL_DMAMUX_GetSyncID

LL_DMAMUX_EnableRequestGen

Function name

__STATIC_INLINE void LL_DMAMUX_EnableRequestGen (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Enable the Request Generator.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_EnableRequestGen

LL_DMAMUX_DisableRequestGen

Function name

__STATIC_INLINE void LL_DMAMUX_DisableRequestGen (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Disable the Request Generator.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_DisableRequestGen

LL_DMAMUX_IsEnabledRequestGen

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledRequestGen (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Check if the Request Generator is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGxCR GE LL_DMAMUX_IsEnabledRequestGen

LL_DMAMUX_SetRequestGenPolarity

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestGenPolarity (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t Polarity)

Function description

Set the polarity of the signal on which the DMA request is generated.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
- **Polarity:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_NO_EVENT
  - LL_DMAMUX_REQ_GEN_POL_RISING
  - LL_DMAMUX_REQ_GEN_POL_FALLING
  - LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR GPOL LL_DMAMUX_SetRequestGenPolarity

LL_DMAMUX_GetRequestGenPolarity

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestGenPolarity (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get the polarity of the signal on which the DMA request is generated.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_GEN_NO_EVENT
  - LL_DMAMUX_REQ_GEN_POL_RISING
  - LL_DMAMUX_REQ_GEN_POL_FALLING
  - LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

Reference Manual to LL API cross reference:

- RGxCR GPOL LL_DMAMUX_GetRequestGenPolarity

LL_DMAMUX_SetGenRequestNb

Function name

__STATIC_INLINE void LL_DMAMUX_SetGenRequestNb (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestNb)

Function description

Set the number of DMA request that will be autorized after a generation event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
- **RequestNb:** This parameter must be a value between Min_Data = 1 and Max_Data = 32.

Return values

- **None:**

Notes

- This field can only be written when Generator is disabled.

Reference Manual to LL API cross reference:

- RGxCR GNBREQ LL_DMAMUX_SetGenRequestNb

LL_DMAMUX_GetGenRequestNb

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetGenRequestNb (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get the number of DMA request that will be autorized after a generation event.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **Between:** Min_Data = 1 and Max_Data = 32

Reference Manual to LL API cross reference:

- RGxCR GNBREQ LL_DMAMUX_GetGenRequestNb

LL_DMAMUX_SetRequestSignalID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestSignalID (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel, uint32_t RequestSignalID)

Function description

Set DMAMUX external Request Signal ID on DMAMUX Request Generation Trigger Event Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3
- **RequestSignalID:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_EXTI_LINE0
  - LL_DMAMUX_REQ_GEN_EXTI_LINE1
  - LL_DMAMUX_REQ_GEN_EXTI_LINE2
  - LL_DMAMUX_REQ_GEN_EXTI_LINE3
  - LL_DMAMUX_REQ_GEN_EXTI_LINE4
  - LL_DMAMUX_REQ_GEN_EXTI_LINE5
  - LL_DMAMUX_REQ_GEN_EXTI_LINE6
  - LL_DMAMUX_REQ_GEN_EXTI_LINE7
  - LL_DMAMUX_REQ_GEN_EXTI_LINE8
  - LL_DMAMUX_REQ_GEN_EXTI_LINE9
  - LL_DMAMUX_REQ_GEN_EXTI_LINE10
  - LL_DMAMUX_REQ_GEN_EXTI_LINE11
  - LL_DMAMUX_REQ_GEN_EXTI_LINE12
  - LL_DMAMUX_REQ_GEN_EXTI_LINE13
  - LL_DMAMUX_REQ_GEN_EXTI_LINE14
  - LL_DMAMUX_REQ_GEN_EXTI_LINE15
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH0
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH1
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH2
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH3 (*)
  - LL_DMAMUX_REQ_GEN_TIM14_OC

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- RGxCR SIG_ID LL_DMAMUX_SetRequestSignalID

LL_DMAMUX_GetRequestSignalID

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_GetRequestSignalID (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Get DMAMUX external Request Signal ID set on DMAMUX Channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_GEN_EXTI_LINE0
  - LL_DMAMUX_REQ_GEN_EXTI_LINE1
  - LL_DMAMUX_REQ_GEN_EXTI_LINE2
  - LL_DMAMUX_REQ_GEN_EXTI_LINE3
  - LL_DMAMUX_REQ_GEN_EXTI_LINE4
  - LL_DMAMUX_REQ_GEN_EXTI_LINE5
  - LL_DMAMUX_REQ_GEN_EXTI_LINE6
  - LL_DMAMUX_REQ_GEN_EXTI_LINE7
  - LL_DMAMUX_REQ_GEN_EXTI_LINE8
  - LL_DMAMUX_REQ_GEN_EXTI_LINE9
  - LL_DMAMUX_REQ_GEN_EXTI_LINE10
  - LL_DMAMUX_REQ_GEN_EXTI_LINE11
  - LL_DMAMUX_REQ_GEN_EXTI_LINE12
  - LL_DMAMUX_REQ_GEN_EXTI_LINE13
  - LL_DMAMUX_REQ_GEN_EXTI_LINE14
  - LL_DMAMUX_REQ_GEN_EXTI_LINE15
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH0
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH1
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH2
  - LL_DMAMUX_REQ_GEN_DMAMUX_CH3 (*)
  - LL_DMAMUX_REQ_GEN_TIM14_OC

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- RGxCR SIG_ID LL_DMAMUX_GetRequestSignalID

LL_DMAMUX_IsActiveFlag_SO0

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO0 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 0.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF0 LL_DMAMUX_IsActiveFlag_SO0

LL_DMAMUX_IsActiveFlag_SO1

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO1 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 1.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF1 LL_DMAMUX_IsActiveFlag_SO1

LL_DMAMUX_IsActiveFlag_SO2

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO2 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 2.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF2 LL_DMAMUX_IsActiveFlag_SO2

LL_DMAMUX_IsActiveFlag_SO3

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO3 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 3.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF3 LL_DMAMUX_IsActiveFlag_SO3

LL_DMAMUX_IsActiveFlag_SO4

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_SO4 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Synchronization Event Overrun Flag Channel 4.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR SOF4 LL_DMAMUX_IsActiveFlag_SO4

LL_DMAMUX_IsActiveFlag_RGO0

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO0 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 0 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF0 LL_DMAMUX_IsActiveFlag_RGO0

LL_DMAMUX_IsActiveFlag_RGO1

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO1 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 1 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF1 LL_DMAMUX_IsActiveFlag_RGO1

LL_DMAMUX_IsActiveFlag_RGO2

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO2 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 2 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF2 LL_DMAMUX_IsActiveFlag_RGO2

LL_DMAMUX_IsActiveFlag_RGO3

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsActiveFlag_RGO3 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Get Request Generator 3 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGSR OF3 LL_DMAMUX_IsActiveFlag_RGO3

LL_DMAMUX_ClearFlag_SO0

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO0 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 0.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF0 LL_DMAMUX_ClearFlag_SO0

LL_DMAMUX_ClearFlag_SO1

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO1 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 1.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF1 LL_DMAMUX_ClearFlag_SO1

LL_DMAMUX_ClearFlag_SO2

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO2 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 2.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF2 LL_DMAMUX_ClearFlag_SO2

LL_DMAMUX_ClearFlag_SO3

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO3 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 3.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF3 LL_DMAMUX_ClearFlag_SO3

LL_DMAMUX_ClearFlag_SO4

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_SO4 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Synchronization Event Overrun Flag Channel 4.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFR CSOF4 LL_DMAMUX_ClearFlag_SO4

LL_DMAMUX_ClearFlag_RGO0

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO0 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 0 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF0 LL_DMAMUX_ClearFlag_RGO0

LL_DMAMUX_ClearFlag_RGO1

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO1 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 1 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF1 LL_DMAMUX_ClearFlag_RGO1

LL_DMAMUX_ClearFlag_RGO2

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO2 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 2 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF2 LL_DMAMUX_ClearFlag_RGO2

LL_DMAMUX_ClearFlag_RGO3

Function name

__STATIC_INLINE void LL_DMAMUX_ClearFlag_RGO3 (const DMAMUX_Channel_TypeDef * DMAMUXx)

Function description

Clear Request Generator 3 Trigger Event Overrun Flag.

Parameters

- **DMAMUXx:** DMAMUXx DMAMUXx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGCFR COF3 LL_DMAMUX_ClearFlag_RGO3

LL_DMAMUX_EnableIT_SO

Function name

__STATIC_INLINE void LL_DMAMUX_EnableIT_SO (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Enable the Synchronization Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_EnableIT_SO

LL_DMAMUX_DisableIT_SO

Function name

__STATIC_INLINE void LL_DMAMUX_DisableIT_SO (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Disable the Synchronization Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_DisableIT_SO

LL_DMAMUX_IsEnabledIT_SO

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_SO (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t Channel)

Function description

Check if the Synchronization Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **Channel:** This parameter can be one of the following values:
  - LL_DMAMUX_CHANNEL_0
  - LL_DMAMUX_CHANNEL_1
  - LL_DMAMUX_CHANNEL_2
  - LL_DMAMUX_CHANNEL_3 (*)
  - LL_DMAMUX_CHANNEL_4 (*)
  - LL_DMAMUX_CHANNEL_5 (*)
  - LL_DMAMUX_CHANNEL_6 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR SOIE LL_DMAMUX_IsEnabledIT_SO

LL_DMAMUX_EnableIT_RGO

Function name

__STATIC_INLINE void LL_DMAMUX_EnableIT_RGO (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Enable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_EnableIT_RGO

LL_DMAMUX_DisableIT_RGO

Function name

__STATIC_INLINE void LL_DMAMUX_DisableIT_RGO (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Disable the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **None:**

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_DisableIT_RGO

LL_DMAMUX_IsEnabledIT_RGO

Function name

__STATIC_INLINE uint32_t LL_DMAMUX_IsEnabledIT_RGO (const DMAMUX_Channel_TypeDef * DMAMUXx, uint32_t RequestGenChannel)

Function description

Check if the Request Generation Trigger Event Overrun Interrupt on DMAMUX channel x is enabled or disabled.

Parameters

- **DMAMUXx:** DMAMUXx Instance
- **RequestGenChannel:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_GEN_0
  - LL_DMAMUX_REQ_GEN_1
  - LL_DMAMUX_REQ_GEN_2
  - LL_DMAMUX_REQ_GEN_3

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RGxCR OIE LL_DMAMUX_IsEnabledIT_RGO

49.2 DMAMUX Firmware driver defines

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

49.2.1 DMAMUX

DMAMUX

DMAMUX Channel

LL_DMAMUX_CHANNEL_0

DMAMUX Channel 0 connected to DMA1 Channel 1

LL_DMAMUX_CHANNEL_1

DMAMUX Channel 1 connected to DMA1 Channel 2

LL_DMAMUX_CHANNEL_2

DMAMUX Channel 2 connected to DMA1 Channel 3

LL_DMAMUX_CHANNEL_3

DMAMUX Channel 3 connected to DMA1 Channel 4

LL_DMAMUX_CHANNEL_4

DMAMUX Channel 4 connected to DMA1 Channel 5

LL_DMAMUX_CHANNEL_5

DMAMUX Channel 5 connected to DMA1 Channel 6

LL_DMAMUX_CHANNEL_6

DMAMUX Channel 6 connected to DMA1 Channel 7

Clear Flags Defines

LL_DMAMUX_CFR_CSOF0

Synchronization Event Overrun Flag Channel 0

LL_DMAMUX_CFR_CSOF1

Synchronization Event Overrun Flag Channel 1

LL_DMAMUX_CFR_CSOF2

Synchronization Event Overrun Flag Channel 2

LL_DMAMUX_CFR_CSOF3

Synchronization Event Overrun Flag Channel 3

LL_DMAMUX_CFR_CSOF4

Synchronization Event Overrun Flag Channel 4

LL_DMAMUX_RGCFR_RGCOF0

Request Generator 0 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF1

Request Generator 1 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF2

Request Generator 2 Trigger Event Overrun Flag

LL_DMAMUX_RGCFR_RGCOF3

Request Generator 3 Trigger Event Overrun Flag

Get Flags Defines

LL_DMAMUX_CSR_SOF0

Synchronization Event Overrun Flag Channel 0

LL_DMAMUX_CSR_SOF1

Synchronization Event Overrun Flag Channel 1

LL_DMAMUX_CSR_SOF2

Synchronization Event Overrun Flag Channel 2

LL_DMAMUX_CSR_SOF3

Synchronization Event Overrun Flag Channel 3

LL_DMAMUX_CSR_SOF4

Synchronization Event Overrun Flag Channel 4

LL_DMAMUX_RGSR_RGOF0

Request Generator 0 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF1

Request Generator 1 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF2

Request Generator 2 Trigger Event Overrun Flag

LL_DMAMUX_RGSR_RGOF3

Request Generator 3 Trigger Event Overrun Flag

IT Defines

LL_DMAMUX_CCR_SOIE

Synchronization Event Overrun Interrupt

LL_DMAMUX_RGCR_RGOIE

Request Generation Trigger Event Overrun Interrupt

Transfer request

LL_DMAMUX_REQ_MEM2MEM

memory to memory transfer

LL_DMAMUX_REQ_GENERATOR0

DMAMUX request generator 0

LL_DMAMUX_REQ_GENERATOR1

DMAMUX request generator 1

LL_DMAMUX_REQ_GENERATOR2

DMAMUX request generator 2

LL_DMAMUX_REQ_GENERATOR3

DMAMUX request generator 3

LL_DMAMUX_REQ_ADC1

DMAMUX ADC1 request

LL_DMAMUX_REQ_I2C1_RX

DMAMUX I2C1 RX request

LL_DMAMUX_REQ_I2C1_TX

DMAMUX I2C1 TX request

LL_DMAMUX_REQ_I2C2_RX

DMAMUX I2C2 RX request

LL_DMAMUX_REQ_I2C2_TX

DMAMUX I2C2 TX request

LL_DMAMUX_REQ_SPI1_RX

DMAMUX SPI1 RX request

LL_DMAMUX_REQ_SPI1_TX

DMAMUX SPI1 TX request

LL_DMAMUX_REQ_SPI2_RX

DMAMUX SPI2 RX request

LL_DMAMUX_REQ_SPI2_TX

DMAMUX SPI2 TX request

LL_DMAMUX_REQ_TIM1_CH1

DMAMUX TIM1 CH1 request

LL_DMAMUX_REQ_TIM1_CH2

DMAMUX TIM1 CH2 request

LL_DMAMUX_REQ_TIM1_CH3

DMAMUX TIM1 CH3 request

LL_DMAMUX_REQ_TIM1_CH4

DMAMUX TIM1 CH4 request

LL_DMAMUX_REQ_TIM1_TRIG_COM

DMAMUX TIM1 TRIG COM request

LL_DMAMUX_REQ_TIM1_UP

DMAMUX TIM1 UP request

LL_DMAMUX_REQ_TIM2_CH1

DMAMUX TIM2 CH1 request

LL_DMAMUX_REQ_TIM2_CH2

DMAMUX TIM2 CH2 request

LL_DMAMUX_REQ_TIM2_CH3

DMAMUX TIM2 CH3 request

LL_DMAMUX_REQ_TIM2_CH4

DMAMUX TIM2 CH4 request

LL_DMAMUX_REQ_TIM2_TRIG

DMAMUX TIM2 TRIG request

LL_DMAMUX_REQ_TIM2_UP

DMAMUX TIM2 UP request

LL_DMAMUX_REQ_TIM3_CH1

DMAMUX TIM3 CH1 request

LL_DMAMUX_REQ_TIM3_CH2

DMAMUX TIM3 CH2 request

LL_DMAMUX_REQ_TIM3_CH3

DMAMUX TIM3 CH3 request

LL_DMAMUX_REQ_TIM3_CH4

DMAMUX TIM3 CH4 request

LL_DMAMUX_REQ_TIM3_TRIG

DMAMUX TIM3 TRIG request

LL_DMAMUX_REQ_TIM3_UP

DMAMUX TIM3 UP request

LL_DMAMUX_REQ_TIM15_CH1

DMAMUX TIM15 CH1 request

LL_DMAMUX_REQ_TIM15_CH2

DMAMUX TIM15 CH2 request

LL_DMAMUX_REQ_TIM15_TRIG_COM

DMAMUX TIM15 TRIG COM request

LL_DMAMUX_REQ_TIM15_UP

DMAMUX TIM15 UP request

LL_DMAMUX_REQ_TIM16_CH1

DMAMUX TIM16 CH1 request

LL_DMAMUX_REQ_TIM16_TRIG_COM

DMAMUX TIM16 TRIG COM request

LL_DMAMUX_REQ_TIM16_UP

DMAMUX TIM16 UP request

LL_DMAMUX_REQ_TIM17_CH1

DMAMUX TIM17 CH1 request

LL_DMAMUX_REQ_TIM17_TRIG_COM

DMAMUX TIM17 TRIG COM request

LL_DMAMUX_REQ_TIM17_UP

DMAMUX TIM17 UP request

LL_DMAMUX_REQ_USART1_RX

DMAMUX USART1 RX request

LL_DMAMUX_REQ_USART1_TX

DMAMUX USART1 TX request

LL_DMAMUX_REQ_USART2_RX

DMAMUX USART2 RX request

LL_DMAMUX_REQ_USART2_TX

DMAMUX USART2 TX request

LL_DMAMUX_REQ_USART3_RX

DMAMUX USART3 RX request

LL_DMAMUX_REQ_USART3_TX

DMAMUX USART3 TX request

LL_DMAMUX_REQ_USART4_RX

DMAMUX USART4 RX request

LL_DMAMUX_REQ_USART4_TX

DMAMUX USART4 TX request

External Request Signal Generation

LL_DMAMUX_REQ_GEN_EXTI_LINE0

Request signal generation from EXTI Line0

LL_DMAMUX_REQ_GEN_EXTI_LINE1

Request signal generation from EXTI Line1

LL_DMAMUX_REQ_GEN_EXTI_LINE2

Request signal generation from EXTI Line2

LL_DMAMUX_REQ_GEN_EXTI_LINE3

Request signal generation from EXTI Line3

LL_DMAMUX_REQ_GEN_EXTI_LINE4

Request signal generation from EXTI Line4

LL_DMAMUX_REQ_GEN_EXTI_LINE5

Request signal generation from EXTI Line5

LL_DMAMUX_REQ_GEN_EXTI_LINE6

Request signal generation from EXTI Line6

LL_DMAMUX_REQ_GEN_EXTI_LINE7

Request signal generation from EXTI Line7

LL_DMAMUX_REQ_GEN_EXTI_LINE8

Request signal generation from EXTI Line8

LL_DMAMUX_REQ_GEN_EXTI_LINE9

Request signal generation from EXTI Line9

LL_DMAMUX_REQ_GEN_EXTI_LINE10

Request signal generation from EXTI Line10

LL_DMAMUX_REQ_GEN_EXTI_LINE11

Request signal generation from EXTI Line11

LL_DMAMUX_REQ_GEN_EXTI_LINE12

Request signal generation from EXTI Line12

LL_DMAMUX_REQ_GEN_EXTI_LINE13

Request signal generation from EXTI Line13

LL_DMAMUX_REQ_GEN_EXTI_LINE14

Request signal generation from EXTI Line14

LL_DMAMUX_REQ_GEN_EXTI_LINE15

Request signal generation from EXTI Line15

LL_DMAMUX_REQ_GEN_DMAMUX_CH0

Request signal generation from DMAMUX channel0 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH1

Request signal generation from DMAMUX channel1 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH2

Request signal generation from DMAMUX channel2 Event

LL_DMAMUX_REQ_GEN_DMAMUX_CH3

Request signal generation from DMAMUX channel3 Event

LL_DMAMUX_REQ_GEN_TIM14_OC

Request signal generation from TIM14 OC

Request Generator Channel

LL_DMAMUX_REQ_GEN_0

LL_DMAMUX_REQ_GEN_1

LL_DMAMUX_REQ_GEN_2

LL_DMAMUX_REQ_GEN_3

External Request Signal Generation Polarity

LL_DMAMUX_REQ_GEN_NO_EVENT

No external DMA request generation

LL_DMAMUX_REQ_GEN_POL_RISING

External DMA request generation on event on rising edge

LL_DMAMUX_REQ_GEN_POL_FALLING

External DMA request generation on event on falling edge

LL_DMAMUX_REQ_GEN_POL_RISING_FALLING

External DMA request generation on rising and falling edge

Synchronization Signal Event

LL_DMAMUX_SYNC_EXTI_LINE0

Synchronization signal from EXTI Line0

LL_DMAMUX_SYNC_EXTI_LINE1

Synchronization signal from EXTI Line1

LL_DMAMUX_SYNC_EXTI_LINE2

Synchronization signal from EXTI Line2

LL_DMAMUX_SYNC_EXTI_LINE3

Synchronization signal from EXTI Line3

LL_DMAMUX_SYNC_EXTI_LINE4

Synchronization signal from EXTI Line4

LL_DMAMUX_SYNC_EXTI_LINE5

Synchronization signal from EXTI Line5

LL_DMAMUX_SYNC_EXTI_LINE6

Synchronization signal from EXTI Line6

LL_DMAMUX_SYNC_EXTI_LINE7

Synchronization signal from EXTI Line7

LL_DMAMUX_SYNC_EXTI_LINE8

Synchronization signal from EXTI Line8

LL_DMAMUX_SYNC_EXTI_LINE9

Synchronization signal from EXTI Line9

LL_DMAMUX_SYNC_EXTI_LINE10

Synchronization signal from EXTI Line10

LL_DMAMUX_SYNC_EXTI_LINE11

Synchronization signal from EXTI Line11

LL_DMAMUX_SYNC_EXTI_LINE12

Synchronization signal from EXTI Line12

LL_DMAMUX_SYNC_EXTI_LINE13

Synchronization signal from EXTI Line1 3

LL_DMAMUX_SYNC_EXTI_LINE14

Synchronization signal from EXTI Line1 4

LL_DMAMUX_SYNC_EXTI_LINE15

Synchronization signal from EXTI Line1 5

LL_DMAMUX_SYNC_DMAMUX_CH0

Synchronization signal from DMAMUX channel0 Event

LL_DMAMUX_SYNC_DMAMUX_CH1

Synchronization signal from DMAMUX channel1 Event

LL_DMAMUX_SYNC_DMAMUX_CH2

Synchronization signal from DMAMUX channel2 Event

LL_DMAMUX_SYNC_DMAMUX_CH3

Synchronization signal from DMAMUX channel3 Event

LL_DMAMUX_SYNC_TIM14_OC

Synchronization signal from TIM14 OC

Synchronization Signal Polarity

LL_DMAMUX_SYNC_NO_EVENT

All requests are blocked

LL_DMAMUX_SYNC_POL_RISING

Synchronization on event on rising edge

LL_DMAMUX_SYNC_POL_FALLING

Synchronization on event on falling edge

LL_DMAMUX_SYNC_POL_RISING_FALLING

Synchronization on event on rising and falling edge

Common Write and read registers macros

LL_DMAMUX_WriteReg

Description:

- Write a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_DMAMUX_ReadReg

Description:

- Read a value in DMAMUX register.

Parameters:

- __INSTANCE__: DMAMUX Instance
- __REG__: Register to be read

Return value:

- Register: value

50 LL DMA Generic Driver

50.1 DMA Firmware driver registers structures

50.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32c0xx_ll_dma.h

Data Fields

- ***uint32_t PeriphOrM2MSrcAddress***
- ***uint32_t MemoryOrM2MDstAddress***
- ***uint32_t Direction***
- ***uint32_t Mode***
- ***uint32_t PeriphOrM2MSrcIncMode***
- ***uint32_t MemoryOrM2MDstIncMode***
- ***uint32_t PeriphOrM2MSrcDataSize***
- ***uint32_t MemoryOrM2MDstDataSize***
- ***uint32_t NbData***
- ***uint32_t PeriphRequest***
- ***uint32_t Priority***

Field Documentation

- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcAddress***
  Specifies the peripheral base address for DMA transfer or as Source base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstAddress***
  Specifies the memory base address for DMA transfer or as Destination base address in case of memory to memory transfer direction.This parameter must be a value between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- ***uint32_t LL_DMA_InitTypeDef::Direction***
  Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of ***DMA_LL_EC_DIRECTION***This feature can be modified afterwards using unitary function **LL_DMA_SetDataTransferDirection()**.
- ***uint32_t LL_DMA_InitTypeDef::Mode***
  Specifies the normal or circular operation mode. This parameter can be a value of ***DMA_LL_EC_MODE***
  Note:
  – : The circular buffer mode cannot be used if the memory to memory data transfer direction is configured on the selected Channel

  This feature can be modified afterwards using unitary function **LL_DMA_SetMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcIncMode***
  Specifies whether the Peripheral address or Source address in case of memory to memory transfer directionis incremented or not. This parameter can be a value of ***DMA_LL_EC_PERIPH***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstIncMode***
  Specifies whether the Memory address or Destination address in case of memory to memory transfer direction is incremented or not. This parameter can be a value of ***DMA_LL_EC_MEMORY***This feature can be modified afterwards using unitary function **LL_DMA_SetMemoryIncMode()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphOrM2MSrcDataSize***
  Specifies the Peripheral data size alignment or Source data size alignment (byte, half word, word)in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_PDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphSize()**.

- ***uint32_t LL_DMA_InitTypeDef::MemoryOrM2MDstDataSize***
  Specifies the Memory data size alignment or Destination data size alignment (byte, half word, word) in case of memory to memory transfer direction. This parameter can be a value of ***DMA_LL_EC_MDATAALIGN***This feature can be modified afterwards using unitary function **LL_DMA_SetMemorySize()**.
- ***uint32_t LL_DMA_InitTypeDef::NbData***
  Specifies the number of data to transfer, in data unit. The data unit is equal to the source buffer configuration set in PeripheralSize or MemorySize parameters depending in the transfer direction. This parameter must be a value between Min_Data = 0 and Max_Data = 0x0000FFFFThis feature can be modified afterwards using unitary function **LL_DMA_SetDataLength()**.
- ***uint32_t LL_DMA_InitTypeDef::PeriphRequest***
  Specifies the peripheral request. This parameter can be a value of ***DMAMUX_LL_EC_REQUEST***This feature can be modified afterwards using unitary function **LL_DMA_SetPeriphRequest()**.
- ***uint32_t LL_DMA_InitTypeDef::Priority***
  Specifies the channel priority level. This parameter can be a value of ***DMA_LL_EC_PRIORITY***This feature can be modified afterwards using unitary function **LL_DMA_SetChannelPriorityLevel()**.

50.2 DMA Firmware driver API description

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

50.2.1 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 DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

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 DMA channel.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

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 DMA channel is enabled or disabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Configuration)

Function description

Configure all parameters link to DMA transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **Configuration:** This parameter must be a combination of all the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH or LL_DMA_DIRECTION_MEMORY_TO_MEMORY
  - LL_DMA_MODE_NORMAL or LL_DMA_MODE_CIRCULAR
  - LL_DMA_PERIPH_INCREMENT or LL_DMA_PERIPH_NOINCREMENT
  - LL_DMA_MEMORY_INCREMENT or LL_DMA_MEMORY_NOINCREMENT
  - LL_DMA_PDATAALIGN_BYTE or LL_DMA_PDATAALIGN_HALFWORD or LL_DMA_PDATAALIGN_WORD
  - LL_DMA_MDATAALIGN_BYTE or LL_DMA_MDATAALIGN_HALFWORD or LL_DMA_MDATAALIGN_WORD
  - LL_DMA_PRIORITY_LOW or LL_DMA_PRIORITY_MEDIUM or LL_DMA_PRIORITY_HIGH or LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR DIR LL_DMA_ConfigTransfer
- CCR MEM2MEM LL_DMA_ConfigTransfer
- CCR CIRC LL_DMA_ConfigTransfer
- CCR PINC LL_DMA_ConfigTransfer
- CCR MINC LL_DMA_ConfigTransfer
- CCR PSIZE LL_DMA_ConfigTransfer
- CCR MSIZE LL_DMA_ConfigTransfer
- CCR PL LL_DMA_ConfigTransfer

LL_DMA_SetDataTransferDirection

Function name

__STATIC_INLINE void LL_DMA_SetDataTransferDirection (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Direction)

Function description

Set Data transfer direction (read from peripheral or from memory).

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR DIR LL_DMA_SetDataTransferDirection
- CCR MEM2MEM 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 Data transfer direction (read from peripheral or from memory).

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR DIR LL_DMA_GetDataTransferDirection
- CCR MEM2MEM LL_DMA_GetDataTransferDirection

LL_DMA_SetMode

Function name

__STATIC_INLINE void LL_DMA_SetMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Mode)

Function description

Set DMA mode circular or normal.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **Mode:** This parameter can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR

Return values

- **None:**

Notes

- The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR CIRC LL_DMA_SetMode

LL_DMA_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get DMA mode circular or normal.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MODE_NORMAL
  - LL_DMA_MODE_CIRCULAR

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR CIRC LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name

__STATIC_INLINE void LL_DMA_SetPeriphIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcIncMode)

Function description

Set Peripheral increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values:
  - LL_DMA_PERIPH_INCREMENT
  - LL_DMA_PERIPH_NOINCREMENT

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR PINC LL_DMA_SetPeriphIncMode

LL_DMA_GetPeriphIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Peripheral increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PERIPH_INCREMENT
  - LL_DMA_PERIPH_NOINCREMENT

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR PINC LL_DMA_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name

__STATIC_INLINE void LL_DMA_SetMemoryIncMode (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstIncMode)

Function description

Set Memory increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **MemoryOrM2MDstIncMode:** This parameter can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR MINC LL_DMA_SetMemoryIncMode

LL_DMA_GetMemoryIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Memory increment mode.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR MINC LL_DMA_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name

__STATIC_INLINE void LL_DMA_SetPeriphSize (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphOrM2MSrcDataSize)

Function description

Set Peripheral size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **PeriphOrM2MSrcDataSize:** This parameter can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Peripheral size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name

__STATIC_INLINE void LL_DMA_SetMemorySize (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryOrM2MDstDataSize)

Function description

Set Memory size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **MemoryOrM2MDstDataSize:** This parameter can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Memory size.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Priority)

Function description

Set Channel priority level.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **Priority:** This parameter can be one of the following values:
  - LL_DMA_PRIORITY_LOW
  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR PL 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_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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PRIORITY_LOW
  - LL_DMA_PRIORITY_MEDIUM
  - LL_DMA_PRIORITY_HIGH
  - LL_DMA_PRIORITY_VERYHIGH

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR PL LL_DMA_GetChannelPriorityLevel

LL_DMA_SetDataLength

Function name

__STATIC_INLINE void LL_DMA_SetDataLength (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t NbData)

Function description

Set Number of data to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF

Return values

- **None:**

Notes

- This action has no effect if channel is enabled.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataLength (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Number of data to transfer.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Once the channel is enabled, the return value indicate the remaining bytes to be transmitted.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CNDTR NDT LL_DMA_GetDataLength

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t SrcAddress, uint32_t DstAddress, uint32_t Direction)

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_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 (*)
- **SrcAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **DstAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF
- **Direction:** This parameter can be one of the following values:
  - LL_DMA_DIRECTION_PERIPH_TO_MEMORY
  - LL_DMA_DIRECTION_MEMORY_TO_PERIPH
  - LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Return values

- **None:**

Notes

- This API must not be called when the DMA channel is enabled.
- Each peripheral using DMA provides an API to get directly the register address (LL_PPP_DMA_GetRegAddr).
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_ConfigAddresses
- CMAR MA LL_DMA_ConfigAddresses

LL_DMA_SetMemoryAddress

Function name

__STATIC_INLINE void LL_DMA_SetMemoryAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA channel is enabled.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name

__STATIC_INLINE void LL_DMA_SetPeriphAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t PeriphAddress)

Function description

Set the Peripheral address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **PeriphAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- This API must not be called when the DMA channel is enabled.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Memory address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get Peripheral address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_PERIPH_TO_MEMORY or LL_DMA_DIRECTION_MEMORY_TO_PERIPH only.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA channel is enabled.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MDstAddress (const DMA_TypeDef * DMAx, uint32_t Channel, uint32_t MemoryAddress)

Function description

Set the Memory to Memory Destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)
- **MemoryAddress:** Between Min_Data = 0 and Max_Data = 0xFFFFFFFF

Return values

- **None:**

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- This API must not be called when the DMA channel is enabled.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get the Memory to Memory Source address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get the Memory to Memory Destination address.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_GetM2MDstAddress

LL_DMA_SetPeriphRequest

Function name

__STATIC_INLINE void LL_DMA_SetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel, uint32_t Request)

Function description

Set DMA request for DMA Channels on DMAMUX Channel x.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX (*)
  - LL_DMAMUX_REQ_I2C2_TX (*)
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX (*)
  - LL_DMAMUX_REQ_SPI2_TX (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_TRIG_COM
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM2_CH1 (*)
  - LL_DMAMUX_REQ_TIM2_CH2 (*)
  - LL_DMAMUX_REQ_TIM2_CH3 (*)
  - LL_DMAMUX_REQ_TIM2_CH4 (*)
  - LL_DMAMUX_REQ_TIM2_TRIG_COM (*)
  - LL_DMAMUX_REQ_TIM2_UP (*)
  - LL_DMAMUX_REQ_TIM3_CH1
  - LL_DMAMUX_REQ_TIM3_CH2
  - LL_DMAMUX_REQ_TIM3_CH3
  - LL_DMAMUX_REQ_TIM3_CH4
  - LL_DMAMUX_REQ_TIM3_TRIG
  - LL_DMAMUX_REQ_TIM3_UP
  - LL_DMAMUX_REQ_TIM16_CH1
  - LL_DMAMUX_REQ_TIM16_TRIG_COM
  - LL_DMAMUX_REQ_TIM16_UP
  - LL_DMAMUX_REQ_TIM17_CH1
  - LL_DMAMUX_REQ_TIM17_TRIG_COM
  - LL_DMAMUX_REQ_TIM17_UP
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX (*)
  - LL_DMAMUX_REQ_USART3_TX (*)
  - LL_DMAMUX_REQ_USART4_RX (*)
  - LL_DMAMUX_REQ_USART4_TX (*)

Return values

- **None:**

Notes

- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMA_SetPeriphRequest

LL_DMA_GetPeriphRequest

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphRequest (DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Get DMA request for DMA Channels on DMAMUX Channel x.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **Returned:** value can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_GENERATOR0
  - LL_DMAMUX_REQ_GENERATOR1
  - LL_DMAMUX_REQ_GENERATOR2
  - LL_DMAMUX_REQ_GENERATOR3
  - LL_DMAMUX_REQ_ADC1
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX (*)
  - LL_DMAMUX_REQ_I2C2_TX (*)
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX (*)
  - LL_DMAMUX_REQ_SPI2_TX (*)
  - LL_DMAMUX_REQ_TIM1_CH1
  - LL_DMAMUX_REQ_TIM1_CH2
  - LL_DMAMUX_REQ_TIM1_CH3
  - LL_DMAMUX_REQ_TIM1_CH4
  - LL_DMAMUX_REQ_TIM1_TRIG_COM
  - LL_DMAMUX_REQ_TIM1_UP
  - LL_DMAMUX_REQ_TIM2_CH1 (*)
  - LL_DMAMUX_REQ_TIM2_CH2 (*)
  - LL_DMAMUX_REQ_TIM2_CH3 (*)
  - LL_DMAMUX_REQ_TIM2_CH4 (*)
  - LL_DMAMUX_REQ_TIM2_TRIG_COM (*)
  - LL_DMAMUX_REQ_TIM2_UP (*)
  - LL_DMAMUX_REQ_TIM3_CH1
  - LL_DMAMUX_REQ_TIM3_CH2
  - LL_DMAMUX_REQ_TIM3_CH3
  - LL_DMAMUX_REQ_TIM3_CH4
  - LL_DMAMUX_REQ_TIM3_TRIG
  - LL_DMAMUX_REQ_TIM3_UP
  - LL_DMAMUX_REQ_TIM16_CH1
  - LL_DMAMUX_REQ_TIM16_TRIG_COM
  - LL_DMAMUX_REQ_TIM16_UP
  - LL_DMAMUX_REQ_TIM17_CH1
  - LL_DMAMUX_REQ_TIM17_TRIG_COM
  - LL_DMAMUX_REQ_TIM17_UP
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_USART2_RX
  - LL_DMAMUX_REQ_USART2_TX
  - LL_DMAMUX_REQ_USART3_RX (*)
  - LL_DMAMUX_REQ_USART3_TX (*)
  - LL_DMAMUX_REQ_USART4_RX (*)
  - LL_DMAMUX_REQ_USART4_TX (*)

Notes

- DMAMUX channel 0 to 6 are mapped to DMA1 channel 1 to 7.
- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMA_GetPeriphRequest

LL_DMA_IsActiveFlag_GI1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI1 (const DMA_TypeDef * DMAx)

Function description

Get Channel 1 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF1 LL_DMA_IsActiveFlag_GI1

LL_DMA_IsActiveFlag_GI2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI2 (const DMA_TypeDef * DMAx)

Function description

Get Channel 2 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF2 LL_DMA_IsActiveFlag_GI2

LL_DMA_IsActiveFlag_GI3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI3 (const DMA_TypeDef * DMAx)

Function description

Get Channel 3 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF3 LL_DMA_IsActiveFlag_GI3

LL_DMA_IsActiveFlag_GI4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI4 (const DMA_TypeDef * DMAx)

Function description

Get Channel 4 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF4 LL_DMA_IsActiveFlag_GI4

LL_DMA_IsActiveFlag_GI5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI5 (const DMA_TypeDef * DMAx)

Function description

Get Channel 5 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF5 LL_DMA_IsActiveFlag_GI5

LL_DMA_IsActiveFlag_GI6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI6 (const DMA_TypeDef * DMAx)

Function description

Get Channel 6 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF6 LL_DMA_IsActiveFlag_GI6

LL_DMA_IsActiveFlag_GI7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI7 (const DMA_TypeDef * DMAx)

Function description

Get Channel 7 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR GIF7 LL_DMA_IsActiveFlag_GI7

LL_DMA_IsActiveFlag_TC1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (const DMA_TypeDef * DMAx)

Function description

Get Channel 1 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF1 LL_DMA_IsActiveFlag_TC1

LL_DMA_IsActiveFlag_TC2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC2 (const DMA_TypeDef * DMAx)

Function description

Get Channel 2 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF2 LL_DMA_IsActiveFlag_TC2

LL_DMA_IsActiveFlag_TC3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC3 (const DMA_TypeDef * DMAx)

Function description

Get Channel 3 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF3 LL_DMA_IsActiveFlag_TC3

LL_DMA_IsActiveFlag_TC4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC4 (const DMA_TypeDef * DMAx)

Function description

Get Channel 4 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF4 LL_DMA_IsActiveFlag_TC4

LL_DMA_IsActiveFlag_TC5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC5 (const DMA_TypeDef * DMAx)

Function description

Get Channel 5 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF5 LL_DMA_IsActiveFlag_TC5

LL_DMA_IsActiveFlag_TC6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC6 (const DMA_TypeDef * DMAx)

Function description

Get Channel 6 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF6 LL_DMA_IsActiveFlag_TC6

LL_DMA_IsActiveFlag_TC7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC7 (const DMA_TypeDef * DMAx)

Function description

Get Channel 7 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TCIF7 LL_DMA_IsActiveFlag_TC7

LL_DMA_IsActiveFlag_HT1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (const DMA_TypeDef * DMAx)

Function description

Get Channel 1 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF1 LL_DMA_IsActiveFlag_HT1

LL_DMA_IsActiveFlag_HT2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT2 (const DMA_TypeDef * DMAx)

Function description

Get Channel 2 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF2 LL_DMA_IsActiveFlag_HT2

LL_DMA_IsActiveFlag_HT3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT3 (const DMA_TypeDef * DMAx)

Function description

Get Channel 3 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF3 LL_DMA_IsActiveFlag_HT3

LL_DMA_IsActiveFlag_HT4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT4 (const DMA_TypeDef * DMAx)

Function description

Get Channel 4 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF4 LL_DMA_IsActiveFlag_HT4

LL_DMA_IsActiveFlag_HT5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT5 (const DMA_TypeDef * DMAx)

Function description

Get Channel 5 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF5 LL_DMA_IsActiveFlag_HT5

LL_DMA_IsActiveFlag_HT6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT6 (const DMA_TypeDef * DMAx)

Function description

Get Channel 6 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF6 LL_DMA_IsActiveFlag_HT6

LL_DMA_IsActiveFlag_HT7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT7 (const DMA_TypeDef * DMAx)

Function description

Get Channel 7 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR HTIF7 LL_DMA_IsActiveFlag_HT7

LL_DMA_IsActiveFlag_TE1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (const DMA_TypeDef * DMAx)

Function description

Get Channel 1 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF1 LL_DMA_IsActiveFlag_TE1

LL_DMA_IsActiveFlag_TE2

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE2 (const DMA_TypeDef * DMAx)

Function description

Get Channel 2 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF2 LL_DMA_IsActiveFlag_TE2

LL_DMA_IsActiveFlag_TE3

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE3 (const DMA_TypeDef * DMAx)

Function description

Get Channel 3 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF3 LL_DMA_IsActiveFlag_TE3

LL_DMA_IsActiveFlag_TE4

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE4 (const DMA_TypeDef * DMAx)

Function description

Get Channel 4 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF4 LL_DMA_IsActiveFlag_TE4

LL_DMA_IsActiveFlag_TE5

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE5 (const DMA_TypeDef * DMAx)

Function description

Get Channel 5 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF5 LL_DMA_IsActiveFlag_TE5

LL_DMA_IsActiveFlag_TE6

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE6 (const DMA_TypeDef * DMAx)

Function description

Get Channel 6 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF6 LL_DMA_IsActiveFlag_TE6

LL_DMA_IsActiveFlag_TE7

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE7 (const DMA_TypeDef * DMAx)

Function description

Get Channel 7 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- ISR TEIF7 LL_DMA_IsActiveFlag_TE7

LL_DMA_ClearFlag_GI1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 1 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC1, LL_DMA_ClearFlag_HT1, LL_DMA_ClearFlag_TE1. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF1 LL_DMA_ClearFlag_GI1

LL_DMA_ClearFlag_GI2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 2 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC2, LL_DMA_ClearFlag_HT2, LL_DMA_ClearFlag_TE2. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF2 LL_DMA_ClearFlag_GI2

LL_DMA_ClearFlag_GI3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 3 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC3, LL_DMA_ClearFlag_HT3, LL_DMA_ClearFlag_TE3. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF3 LL_DMA_ClearFlag_GI3

LL_DMA_ClearFlag_GI4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 4 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC4, LL_DMA_ClearFlag_HT4, LL_DMA_ClearFlag_TE4. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF4 LL_DMA_ClearFlag_GI4

LL_DMA_ClearFlag_GI5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 5 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC5, LL_DMA_ClearFlag_HT5, LL_DMA_ClearFlag_TE5. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF5 LL_DMA_ClearFlag_GI5

LL_DMA_ClearFlag_GI6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 6 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC6, LL_DMA_ClearFlag_HT6, LL_DMA_ClearFlag_TE6. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF6 LL_DMA_ClearFlag_GI6

LL_DMA_ClearFlag_GI7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Notes

- Do not Clear Channel 7 global interrupt flag when the channel in ON. Instead clear specific flags transfer complete, half transfer & transfer error flag with LL_DMA_ClearFlag_TC7, LL_DMA_ClearFlag_HT7, LL_DMA_ClearFlag_TE7. bug id 2.4.1 in Product Errata Sheet.

Reference Manual to LL API cross reference:

- IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_TC1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF1 LL_DMA_ClearFlag_TC1

LL_DMA_ClearFlag_TC2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF2 LL_DMA_ClearFlag_TC2

LL_DMA_ClearFlag_TC3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF3 LL_DMA_ClearFlag_TC3

LL_DMA_ClearFlag_TC4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF4 LL_DMA_ClearFlag_TC4

LL_DMA_ClearFlag_TC5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF5 LL_DMA_ClearFlag_TC5

LL_DMA_ClearFlag_TC6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF6 LL_DMA_ClearFlag_TC6

LL_DMA_ClearFlag_TC7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF7 LL_DMA_ClearFlag_TC7

LL_DMA_ClearFlag_HT1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF1 LL_DMA_ClearFlag_HT1

LL_DMA_ClearFlag_HT2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF2 LL_DMA_ClearFlag_HT2

LL_DMA_ClearFlag_HT3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF3 LL_DMA_ClearFlag_HT3

LL_DMA_ClearFlag_HT4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF4 LL_DMA_ClearFlag_HT4

LL_DMA_ClearFlag_HT5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF5 LL_DMA_ClearFlag_HT5

LL_DMA_ClearFlag_HT6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF6 LL_DMA_ClearFlag_HT6

LL_DMA_ClearFlag_HT7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF7 LL_DMA_ClearFlag_HT7

LL_DMA_ClearFlag_TE1

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE1 (DMA_TypeDef * DMAx)

Function description

Clear Channel 1 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF1 LL_DMA_ClearFlag_TE1

LL_DMA_ClearFlag_TE2

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE2 (DMA_TypeDef * DMAx)

Function description

Clear Channel 2 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF2 LL_DMA_ClearFlag_TE2

LL_DMA_ClearFlag_TE3

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE3 (DMA_TypeDef * DMAx)

Function description

Clear Channel 3 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF3 LL_DMA_ClearFlag_TE3

LL_DMA_ClearFlag_TE4

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE4 (DMA_TypeDef * DMAx)

Function description

Clear Channel 4 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF4 LL_DMA_ClearFlag_TE4

LL_DMA_ClearFlag_TE5

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE5 (DMA_TypeDef * DMAx)

Function description

Clear Channel 5 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF5 LL_DMA_ClearFlag_TE5

LL_DMA_ClearFlag_TE6

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE6 (DMA_TypeDef * DMAx)

Function description

Clear Channel 6 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF6 LL_DMA_ClearFlag_TE6

LL_DMA_ClearFlag_TE7

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE7 (DMA_TypeDef * DMAx)

Function description

Clear Channel 7 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF7 LL_DMA_ClearFlag_TE7

LL_DMA_EnableIT_TC

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TC (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_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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_EnableIT_TC

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 interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_EnableIT_HT

LL_DMA_EnableIT_TE

Function name

__STATIC_INLINE void LL_DMA_EnableIT_TE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Enable Transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR TEIE LL_DMA_EnableIT_TE

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_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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_DisableIT_TC

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 interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_DisableIT_HT

LL_DMA_DisableIT_TE

Function name

__STATIC_INLINE void LL_DMA_DisableIT_TE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Disable Transfer error interrupt.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR TEIE LL_DMA_DisableIT_TE

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_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 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR TCIE LL_DMA_IsEnabledIT_TC

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 Interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR HTIE LL_DMA_IsEnabledIT_HT

LL_DMA_IsEnabledIT_TE

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledIT_TE (const DMA_TypeDef * DMAx, uint32_t Channel)

Function description

Check if Transfer error Interrupt is enabled.

Parameters

- **DMAx:** DMAx Instance
- **Channel:** This parameter can be one of the following values:
  - 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 (*)

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CCR TEIE LL_DMA_IsEnabledIT_TE

LL_DMA_Init

Function name

ErrorStatus 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_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 (*)
- **DMA_InitStruct:** pointer to a LL_DMA_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are initialized
  - ERROR: Not applicable

Notes

- To convert DMAx_Channely Instance to DMAx Instance and Channely, use helper macros : __LL_DMA_GET_INSTANCE __LL_DMA_GET_CHANNEL
- (*) Availability depends on devices

LL_DMA_DeInit

Function name

ErrorStatus 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_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_ALL

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: DMA registers are de-initialized
  - ERROR: DMA registers are not de-initialized

Notes

- (*) Availability depends on devices

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:**

50.3 DMA Firmware driver defines

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

50.3.1 DMA

DMA

CHANNEL

LL_DMA_CHANNEL_1

DMA Channel 1

LL_DMA_CHANNEL_2

DMA Channel 2

LL_DMA_CHANNEL_3

DMA Channel 3

LL_DMA_CHANNEL_4

DMA Channel 4

LL_DMA_CHANNEL_5

DMA Channel 5

LL_DMA_CHANNEL_6

DMA Channel 6

LL_DMA_CHANNEL_7

DMA Channel 7

LL_DMA_CHANNEL_ALL

DMA Channel all (used only for function

Clear Flags Defines

LL_DMA_IFCR_CGIF1

Channel 1 global flag

LL_DMA_IFCR_CTCIF1

Channel 1 transfer complete flag

LL_DMA_IFCR_CHTIF1

Channel 1 half transfer flag

LL_DMA_IFCR_CTEIF1

Channel 1 transfer error flag

LL_DMA_IFCR_CGIF2

Channel 2 global flag

LL_DMA_IFCR_CTCIF2

Channel 2 transfer complete flag

LL_DMA_IFCR_CHTIF2

Channel 2 half transfer flag

LL_DMA_IFCR_CTEIF2

Channel 2 transfer error flag

LL_DMA_IFCR_CGIF3

Channel 3 global flag

LL_DMA_IFCR_CTCIF3

Channel 3 transfer complete flag

LL_DMA_IFCR_CHTIF3

Channel 3 half transfer flag

LL_DMA_IFCR_CTEIF3

Channel 3 transfer error flag

LL_DMA_IFCR_CGIF4

Channel 4 global flag

LL_DMA_IFCR_CTCIF4

Channel 4 transfer complete flag

LL_DMA_IFCR_CHTIF4

Channel 4 half transfer flag

LL_DMA_IFCR_CTEIF4

Channel 4 transfer error flag

LL_DMA_IFCR_CGIF5

Channel 5 global flag

LL_DMA_IFCR_CTCIF5

Channel 5 transfer complete flag

LL_DMA_IFCR_CHTIF5

Channel 5 half transfer flag

LL_DMA_IFCR_CTEIF5

Channel 5 transfer error flag

LL_DMA_IFCR_CGIF6

Channel 6 global flag

LL_DMA_IFCR_CTCIF6

Channel 6 transfer complete flag

LL_DMA_IFCR_CHTIF6

Channel 6 half transfer flag

LL_DMA_IFCR_CTEIF6

Channel 6 transfer error flag

LL_DMA_IFCR_CGIF7

Channel 7 global flag

LL_DMA_IFCR_CTCIF7

Channel 7 transfer complete flag

LL_DMA_IFCR_CHTIF7

Channel 7 half transfer flag

LL_DMA_IFCR_CTEIF7

Channel 7 transfer error flag

Transfer Direction

LL_DMA_DIRECTION_PERIPH_TO_MEMORY

Peripheral to memory direction

LL_DMA_DIRECTION_MEMORY_TO_PERIPH

Memory to peripheral direction

LL_DMA_DIRECTION_MEMORY_TO_MEMORY

Memory to memory direction

Get Flags Defines

LL_DMA_ISR_GIF1

Channel 1 global flag

LL_DMA_ISR_TCIF1

Channel 1 transfer complete flag

LL_DMA_ISR_HTIF1

Channel 1 half transfer flag

LL_DMA_ISR_TEIF1

Channel 1 transfer error flag

LL_DMA_ISR_GIF2

Channel 2 global flag

LL_DMA_ISR_TCIF2

Channel 2 transfer complete flag

LL_DMA_ISR_HTIF2

Channel 2 half transfer flag

LL_DMA_ISR_TEIF2

Channel 2 transfer error flag

LL_DMA_ISR_GIF3

Channel 3 global flag

LL_DMA_ISR_TCIF3

Channel 3 transfer complete flag

LL_DMA_ISR_HTIF3

Channel 3 half transfer flag

LL_DMA_ISR_TEIF3

Channel 3 transfer error flag

LL_DMA_ISR_GIF4

Channel 4 global flag

LL_DMA_ISR_TCIF4

Channel 4 transfer complete flag

LL_DMA_ISR_HTIF4

Channel 4 half transfer flag

LL_DMA_ISR_TEIF4

Channel 4 transfer error flag

LL_DMA_ISR_GIF5

Channel 5 global flag

LL_DMA_ISR_TCIF5

Channel 5 transfer complete flag

LL_DMA_ISR_HTIF5

Channel 5 half transfer flag

LL_DMA_ISR_TEIF5

Channel 5 transfer error flag

LL_DMA_ISR_GIF6

Channel 6 global flag

LL_DMA_ISR_TCIF6

Channel 6 transfer complete flag

LL_DMA_ISR_HTIF6

Channel 6 half transfer flag

LL_DMA_ISR_TEIF6

Channel 6 transfer error flag

LL_DMA_ISR_GIF7

Channel 7 global flag

LL_DMA_ISR_TCIF7

Channel 7 transfer complete flag

LL_DMA_ISR_HTIF7

Channel 7 half transfer flag

LL_DMA_ISR_TEIF7

Channel 7 transfer error flag

IT Defines

LL_DMA_CCR_TCIE

Transfer complete interrupt

LL_DMA_CCR_HTIE

Half Transfer interrupt

LL_DMA_CCR_TEIE

Transfer error interrupt

Memory data alignment

LL_DMA_MDATAALIGN_BYTE

Memory data alignment : Byte

LL_DMA_MDATAALIGN_HALFWORD

Memory data alignment : HalfWord

LL_DMA_MDATAALIGN_WORD

Memory data alignment : Word

Memory increment mode

LL_DMA_MEMORY_INCREMENT

Memory increment mode Enable

LL_DMA_MEMORY_NOINCREMENT

Memory increment mode Disable

Transfer mode

LL_DMA_MODE_NORMAL

Normal Mode

LL_DMA_MODE_CIRCULAR

Circular Mode

Peripheral data alignment

LL_DMA_PDATAALIGN_BYTE

Peripheral data alignment : Byte

LL_DMA_PDATAALIGN_HALFWORD

Peripheral data alignment : HalfWord

LL_DMA_PDATAALIGN_WORD

Peripheral data alignment : Word

Peripheral increment mode

LL_DMA_PERIPH_INCREMENT

Peripheral increment mode Enable

LL_DMA_PERIPH_NOINCREMENT

Peripheral increment mode Disable

Transfer Priority level

LL_DMA_PRIORITY_LOW

Priority level : Low

LL_DMA_PRIORITY_MEDIUM

Priority level : Medium

LL_DMA_PRIORITY_HIGH

Priority level : High

LL_DMA_PRIORITY_VERYHIGH

Priority level : Very_High

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

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

51 LL EXTI Generic Driver

51.1 EXTI Firmware driver registers structures

51.1.1 LL_EXTI_InitTypeDef

LL_EXTI_InitTypeDef is defined in the stm32c0xx_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***.

51.2 EXTI Firmware driver API description

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

51.2.1 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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_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

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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_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_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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_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

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

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

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

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

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

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

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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_31

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_19
  - LL_EXTI_LINE_23
  - LL_EXTI_LINE_25
  - LL_EXTI_LINE_31

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

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

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

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

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:
  - EXTI CONFIG PORT
- **Line:** This parameter can be one of the following values:
  - LL_EXTI_CONFIG_LINE0
  - LL_EXTI_CONFIG_LINE1
  - LL_EXTI_CONFIG_LINE2
  - LL_EXTI_CONFIG_LINE3
  - LL_EXTI_CONFIG_LINE4
  - LL_EXTI_CONFIG_LINE5
  - LL_EXTI_CONFIG_LINE6
  - LL_EXTI_CONFIG_LINE7
  - LL_EXTI_CONFIG_LINE8
  - LL_EXTI_CONFIG_LINE9
  - LL_EXTI_CONFIG_LINE10
  - LL_EXTI_CONFIG_LINE11
  - LL_EXTI_CONFIG_LINE12
  - LL_EXTI_CONFIG_LINE13
  - LL_EXTI_CONFIG_LINE14
  - LL_EXTI_CONFIG_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_CONFIG_LINE0
  - LL_EXTI_CONFIG_LINE1
  - LL_EXTI_CONFIG_LINE2
  - LL_EXTI_CONFIG_LINE3
  - LL_EXTI_CONFIG_LINE4
  - LL_EXTI_CONFIG_LINE5
  - LL_EXTI_CONFIG_LINE6
  - LL_EXTI_CONFIG_LINE7
  - LL_EXTI_CONFIG_LINE8
  - LL_EXTI_CONFIG_LINE9
  - LL_EXTI_CONFIG_LINE10
  - LL_EXTI_CONFIG_LINE11
  - LL_EXTI_CONFIG_LINE12
  - LL_EXTI_CONFIG_LINE13
  - LL_EXTI_CONFIG_LINE14
  - LL_EXTI_CONFIG_LINE15

Return values

- **Returned:** value can be one of the following values:
  - EXTI CONFIG PORT

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_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:**

51.3 EXTI Firmware driver defines

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

51.3.1 EXTI

EXTI

EXTI CONFIG LINE

LL_EXTI_CONFIG_LINE0

EXTI_POSITION_0 | EXTICR[0]

LL_EXTI_CONFIG_LINE1

EXTI_POSITION_8 | EXTICR[0]

LL_EXTI_CONFIG_LINE2

EXTI_POSITION_16 | EXTICR[0]

LL_EXTI_CONFIG_LINE3

EXTI_POSITION_24 | EXTICR[0]

LL_EXTI_CONFIG_LINE4

EXTI_POSITION_0 | EXTICR[1]

LL_EXTI_CONFIG_LINE5

EXTI_POSITION_8 | EXTICR[1]

LL_EXTI_CONFIG_LINE6

EXTI_POSITION_16 | EXTICR[1]

LL_EXTI_CONFIG_LINE7

EXTI_POSITION_24 | EXTICR[1]

LL_EXTI_CONFIG_LINE8

EXTI_POSITION_0 | EXTICR[2]

LL_EXTI_CONFIG_LINE9

EXTI_POSITION_8 | EXTICR[2]

LL_EXTI_CONFIG_LINE10

EXTI_POSITION_16 | EXTICR[2]

LL_EXTI_CONFIG_LINE11

EXTI_POSITION_24 | EXTICR[2]

LL_EXTI_CONFIG_LINE12

EXTI_POSITION_0 | EXTICR[3]

LL_EXTI_CONFIG_LINE13

EXTI_POSITION_8 | EXTICR[3]

LL_EXTI_CONFIG_LINE14

EXTI_POSITION_16 | EXTICR[3]

LL_EXTI_CONFIG_LINE15

EXTI_POSITION_24 | EXTICR[3]

EXTI CONFIG PORT

LL_EXTI_CONFIG_PORTA

EXTI PORT A

LL_EXTI_CONFIG_PORTB

EXTI PORT B

LL_EXTI_CONFIG_PORTC

EXTI PORT C

LL_EXTI_CONFIG_PORTD

EXTI PORT D

LL_EXTI_CONFIG_PORTF

EXTI PORT F

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_19

Extended line 19

LL_EXTI_LINE_23

Extended line 23

LL_EXTI_LINE_25

Extended line 25

LL_EXTI_LINE_31

Extended line 31

LL_EXTI_LINE_ALL_0_31

All Extended line

LL_EXTI_LINE_ALL_32_63

only line 34 & line 36

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

52 LL GPIO Generic Driver

52.1 GPIO Firmware driver registers structures

52.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32c0xx_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()**.

52.2 GPIO Firmware driver API description

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

52.2.1 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

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_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:**

52.3 GPIO Firmware driver defines

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

52.3.1 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

53 LL I2C Generic Driver

53.1 I2C Firmware driver registers structures

53.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32c0xx_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()**.

53.2 I2C Firmware driver API description

The following section lists the various functions of the I2C library.

53.2.1 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_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_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:**

53.3 I2C Firmware driver defines

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

53.3.1 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.

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

54 LL IWDG Generic Driver

54.1 IWDG Firmware driver API description

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

54.1.1 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

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

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_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_IsReady

Function name

__STATIC_INLINE uint32_t LL_IWDG_IsReady (const IWDG_TypeDef * IWDGx)

Function description

Check if all flags Prescaler, Reload & Window 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

54.2 IWDG Firmware driver defines

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

54.2.1 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

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

55 LL PWR Generic Driver

55.1 PWR Firmware driver API description

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

55.1.1 Detailed description of functions

LL_PWR_EnableFlashPowerDownInSleep

Function name

__STATIC_INLINE void LL_PWR_EnableFlashPowerDownInSleep (void)

Function description

Enable Flash Power-down mode during sleep mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FPD_SLP LL_PWR_EnableFlashPowerDownInSleep

LL_PWR_DisableFlashPowerDownInSleep

Function name

__STATIC_INLINE void LL_PWR_DisableFlashPowerDownInSleep (void)

Function description

Disable Flash Power-down mode during sleep mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FPD_SLP LL_PWR_DisableFlashPowerDownInSleep

LL_PWR_IsEnableFlashPowerDownInSleep

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnableFlashPowerDownInSleep (void)

Function description

Check if flash power-down mode during sleep mode domain is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FPD_SLP LL_PWR_IsEnableFlashPowerDownInSleep

LL_PWR_EnableFlashPowerDownInStop

Function name

__STATIC_INLINE void LL_PWR_EnableFlashPowerDownInStop (void)

Function description

Enable Flash Power-down mode during stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FPD_STOP LL_PWR_EnableFlashPowerDownInStop

LL_PWR_DisableFlashPowerDownInStop

Function name

__STATIC_INLINE void LL_PWR_DisableFlashPowerDownInStop (void)

Function description

Disable Flash Power-down mode during stop mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FPD_STOP LL_PWR_DisableFlashPowerDownInStop

LL_PWR_IsEnableFlashPowerDownInStop

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnableFlashPowerDownInStop (void)

Function description

Check if flash power-down mode during stop mode domain is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR1 FPD_STOP LL_PWR_IsEnableFlashPowerDownInStop

LL_PWR_SetPowerMode

Function name

__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t LowPowerMode)

Function description

Set Low-Power mode.

Parameters

- **LowPowerMode:** This parameter can be one of the following values:
  - LL_PWR_MODE_STOP0
  - LL_PWR_MODE_STANDBY
  - LL_PWR_MODE_SHUTDOWN

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 Low-Power mode.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_MODE_STOP0
  - LL_PWR_MODE_STANDBY
  - LL_PWR_MODE_SHUTDOWN

Reference Manual to LL API cross reference:

- CR1 LPMS LL_PWR_GetPowerMode

LL_PWR_EnableInternWU

Function name

__STATIC_INLINE void LL_PWR_EnableInternWU (void)

Function description

Enable Internal Wake-up line.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWF LL_PWR_EnableInternWU

LL_PWR_DisableInternWU

Function name

__STATIC_INLINE void LL_PWR_DisableInternWU (void)

Function description

Disable Internal Wake-up line.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWF LL_PWR_DisableInternWU

LL_PWR_IsEnabledInternWU

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledInternWU (void)

Function description

Check if Internal Wake-up line is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 EIWF LL_PWR_IsEnabledInternWU

LL_PWR_EnablePUPDCfg

Function name

__STATIC_INLINE void LL_PWR_EnablePUPDCfg (void)

Function description

Enable pull-up and pull-down configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 APC LL_PWR_EnablePUPDCfg

LL_PWR_DisablePUPDCfg

Function name

__STATIC_INLINE void LL_PWR_DisablePUPDCfg (void)

Function description

Disable pull-up and pull-down configuration.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 APC LL_PWR_DisablePUPDCfg

LL_PWR_IsEnabledPUPDCfg

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPUPDCfg (void)

Function description

Check if pull-up and pull-down configuration is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR3 APC LL_PWR_IsEnabledPUPDCfg

LL_PWR_EnableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable the WakeUp PINx functionality.

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

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CR3 EWUP1 LL_PWR_EnableWakeUpPin
- CR3 EWUP2 LL_PWR_EnableWakeUpPin
- CR3 EWUP3 LL_PWR_EnableWakeUpPin
- CR3 EWUP4 LL_PWR_EnableWakeUpPin
- CR3 EWUP5 LL_PWR_EnableWakeUpPin (*)
- CR3 EWUP6 LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable the WakeUp PINx functionality.

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

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CR3 EWUP1 LL_PWR_DisableWakeUpPin
- CR3 EWUP2 LL_PWR_DisableWakeUpPin
- CR3 EWUP3 LL_PWR_DisableWakeUpPin
- CR3 EWUP4 LL_PWR_DisableWakeUpPin
- CR3 EWUP5 LL_PWR_DisableWakeUpPin (*)
- CR3 EWUP6 LL_PWR_DisableWakeUpPin

LL_PWR_IsEnabledWakeUpPin

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWakeUpPin (uint32_t WakeUpPin)

Function description

Check if the WakeUp PINx functionality 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

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CR3 EWUP1 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP2 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP3 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP4 LL_PWR_IsEnabledWakeUpPin
- CR3 EWUP5 LL_PWR_IsEnabledWakeUpPin (*)
- CR3 EWUP6 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 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

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CR4 WP1 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP2 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP3 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP4 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WP5 LL_PWR_SetWakeUpPinPolarityLow (*)
- CR4 WP6 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 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

Return values

- **None:**

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CR4 WP1 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP2 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP3 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP4 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WP5 LL_PWR_SetWakeUpPinPolarityHigh (*)
- CR4 WP6 LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_IsWakeUpPinPolarityLow

Function name

__STATIC_INLINE uint32_t LL_PWR_IsWakeUpPinPolarityLow (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

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Availability depends on devices

Reference Manual to LL API cross reference:

- CR4 WP1 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP2 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP3 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP4 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WP5 LL_PWR_IsWakeUpPinPolarityLow (*)
- CR4 WP6 LL_PWR_IsWakeUpPinPolarityLow

LL_PWR_EnableGPIOPullUp

Function name

__STATIC_INLINE void LL_PWR_EnableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D (*)
  - LL_PWR_GPIO_F
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Notes

- (*) Port D availability depends on devices
- (*) Port D availability depends on devices

Reference Manual to LL API cross reference:

- PUCRA PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRB PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRC PU0-15 LL_PWR_EnableGPIOPullUp
- PUCRD PU0-15 LL_PWR_EnableGPIOPullUp (*)
- PUCRF PU0-15 LL_PWR_EnableGPIOPullUp

LL_PWR_DisableGPIOPullUp

Function name

__STATIC_INLINE void LL_PWR_DisableGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-up state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D (*)
  - LL_PWR_GPIO_F
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Notes

- (*) Port D availability depends on devices

Reference Manual to LL API cross reference:

- PUCRA PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRB PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRC PU0-15 LL_PWR_DisableGPIOPullUp
- PUCRD PU0-15 LL_PWR_DisableGPIOPullUp (*)
- PUCRF PU0-15 LL_PWR_DisableGPIOPullUp

LL_PWR_IsEnabledGPIOPullUp

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullUp (uint32_t GPIO, uint32_t GPIONumber)

Function description

Check if GPIO pull-up state is enabled.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D (*)
  - LL_PWR_GPIO_F
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Port D availability depends on devices
- (*) Port D availability depends on devices

Reference Manual to LL API cross reference:

- PUCRA PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRB PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRC PU0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRD PU0-15 LL_PWR_IsEnabledGPIOPullUp (*)
- PUCRF PU0-15 LL_PWR_IsEnabledGPIOPullUp

LL_PWR_EnableGPIOPullDown

Function name

__STATIC_INLINE void LL_PWR_EnableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Enable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D (*)
  - LL_PWR_GPIO_F
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Notes

- (*) Port D availability depends on devices

Reference Manual to LL API cross reference:

- PDCRA PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRB PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRC PD0-15 LL_PWR_EnableGPIOPullDown
- PDCRD PD0-15 LL_PWR_EnableGPIOPullDown (*)
- PDCRF PD0-15 LL_PWR_EnableGPIOPullDown

LL_PWR_DisableGPIOPullDown

Function name

__STATIC_INLINE void LL_PWR_DisableGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Disable GPIO pull-down state in Standby and Shutdown modes.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D (*)
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **None:**

Notes

- (*) Port D availability depends on devices
- (*) Port D availability depends on devices LL_PWR_GPIO_F

Reference Manual to LL API cross reference:

- PDCRA PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRB PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRC PD0-15 LL_PWR_DisableGPIOPullDown
- PDCRD PD0-15 LL_PWR_DisableGPIOPullDown (*)
- PDCRF PD0-15 LL_PWR_DisableGPIOPullDown

LL_PWR_IsEnabledGPIOPullDown

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Check if GPIO pull-down state is enabled.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
  - LL_PWR_GPIO_C
  - LL_PWR_GPIO_D (*)
  - LL_PWR_GPIO_F
- **GPIONumber:** This parameter can be one of the following values:
  - LL_PWR_GPIO_BIT_0
  - LL_PWR_GPIO_BIT_1
  - LL_PWR_GPIO_BIT_2
  - LL_PWR_GPIO_BIT_3
  - LL_PWR_GPIO_BIT_4
  - LL_PWR_GPIO_BIT_5
  - LL_PWR_GPIO_BIT_6
  - LL_PWR_GPIO_BIT_7
  - LL_PWR_GPIO_BIT_8
  - LL_PWR_GPIO_BIT_9
  - LL_PWR_GPIO_BIT_10
  - LL_PWR_GPIO_BIT_11
  - LL_PWR_GPIO_BIT_12
  - LL_PWR_GPIO_BIT_13
  - LL_PWR_GPIO_BIT_14
  - LL_PWR_GPIO_BIT_15

Return values

- **State:** of bit (1 or 0).

Notes

- (*) Port D availability depends on devices PDCRF PD0-15 LL_PWR_IsEnabledGPIOPullDown
- (*) Port D availability depends on devices

Reference Manual to LL API cross reference:

- PDCRA PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRB PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRC PD0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRD PD0-15 LL_PWR_IsEnabledGPIOPullDown (*)
-

LL_PWR_BKP_SetRegister

Function name

__STATIC_INLINE void LL_PWR_BKP_SetRegister (uint32_t BackupRegister, uint16_t Data)

Function description

Writes a data in a specified Backup data register.

Parameters

- **BackupRegister:** This parameter can be one of the following values:
  - LL_PWR_BKP_DR0
  - LL_PWR_BKP_DR1
  - LL_PWR_BKP_DR2
  - LL_PWR_BKP_DR3
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- PWR_BKPREGx BKP LL_PWR_BKP_SetRegister

LL_PWR_BKP_GetRegister

Function name

__STATIC_INLINE uint32_t LL_PWR_BKP_GetRegister (uint32_t BackupRegister)

Function description

Reads data from the specified Backup data Register.

Parameters

- **BackupRegister:** This parameter can be one of the following values:
  - LL_PWR_BKP_DR0
  - LL_PWR_BKP_DR1
  - LL_PWR_BKP_DR2
  - LL_PWR_BKP_DR3

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- PWR_BKPREGx BKP LL_PWR_BKP_GetRegister

LL_PWR_IsActiveFlag_InternWU

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_InternWU (void)

Function description

Get Internal Wake-up line Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUFI LL_PWR_IsActiveFlag_InternWU

LL_PWR_IsActiveFlag_SB

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_SB (void)

Function description

Get Standby Flag.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 SBF LL_PWR_IsActiveFlag_SB

LL_PWR_IsActiveFlag_WU6

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU6 (void)

Function description

Get Wake-up Flag 6.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF6 LL_PWR_IsActiveFlag_WU6

LL_PWR_IsActiveFlag_WU5

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU5 (void)

Function description

Get Wake-up Flag 5.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF5 LL_PWR_IsActiveFlag_WU5

LL_PWR_IsActiveFlag_WU4

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU4 (void)

Function description

Get Wake-up Flag 4.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF4 LL_PWR_IsActiveFlag_WU4

LL_PWR_IsActiveFlag_WU3

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU3 (void)

Function description

Get Wake-up Flag 3.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF3 LL_PWR_IsActiveFlag_WU3

LL_PWR_IsActiveFlag_WU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU2 (void)

Function description

Get Wake-up Flag 2.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF2 LL_PWR_IsActiveFlag_WU2

LL_PWR_IsActiveFlag_WU1

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU1 (void)

Function description

Get Wake-up Flag 1.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR1 WUF1 LL_PWR_IsActiveFlag_WU1

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:

- SCR CSBF LL_PWR_ClearFlag_SB

LL_PWR_ClearFlag_WU

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU (void)

Function description

Clear Wake-up Flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SCR CWUF LL_PWR_ClearFlag_WU

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:

- SCR CWUF6 LL_PWR_ClearFlag_WU6

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:

- SCR CWUF5 LL_PWR_ClearFlag_WU5

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:

- SCR CWUF4 LL_PWR_ClearFlag_WU4

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:

- SCR CWUF3 LL_PWR_ClearFlag_WU3

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:

- SCR CWUF2 LL_PWR_ClearFlag_WU2

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:

- SCR CWUF1 LL_PWR_ClearFlag_WU1

LL_PWR_IsActiveFlag_FLASH_RDY

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_FLASH_RDY (void)

Function description

Indicate whether or not the flash is ready to be accessed.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR2 FLASH_RDY LL_PWR_IsActiveFlag_FLASH_RDY

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

55.2 PWR Firmware driver defines

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

55.2.1 PWR

PWR

BACKUP

LL_PWR_BKP_NUMBER

LL_PWR_BKP_DR0

LL_PWR_BKP_DR1

LL_PWR_BKP_DR2

LL_PWR_BKP_DR3

Clear Flags Defines

LL_PWR_SCR_CSBF

LL_PWR_SCR_CWUF

LL_PWR_SCR_CWUF6

LL_PWR_SCR_CWUF5

LL_PWR_SCR_CWUF4

LL_PWR_SCR_CWUF3

LL_PWR_SCR_CWUF2

LL_PWR_SCR_CWUF1

Get Flags Defines

LL_PWR_SR1_WUFI

LL_PWR_SR1_SBF

LL_PWR_SR1_WUF6

LL_PWR_SR1_WUF5

LL_PWR_SR1_WUF4

LL_PWR_SR1_WUF3

LL_PWR_SR1_WUF2

LL_PWR_SR1_WUF1

GPIO

LL_PWR_GPIO_A

LL_PWR_GPIO_B

LL_PWR_GPIO_C

LL_PWR_GPIO_D

LL_PWR_GPIO_F

GPIO BIT

LL_PWR_GPIO_BIT_0

LL_PWR_GPIO_BIT_1

LL_PWR_GPIO_BIT_2

LL_PWR_GPIO_BIT_3

LL_PWR_GPIO_BIT_4

LL_PWR_GPIO_BIT_5

LL_PWR_GPIO_BIT_6

LL_PWR_GPIO_BIT_7

LL_PWR_GPIO_BIT_8

LL_PWR_GPIO_BIT_9

LL_PWR_GPIO_BIT_10

LL_PWR_GPIO_BIT_11

LL_PWR_GPIO_BIT_12

LL_PWR_GPIO_BIT_13

LL_PWR_GPIO_BIT_14

LL_PWR_GPIO_BIT_15

MODE PWR

LL_PWR_MODE_STOP0

LL_PWR_MODE_STANDBY

LL_PWR_MODE_SHUTDOWN

WAKEUP

LL_PWR_WAKEUP_PIN1

LL_PWR_WAKEUP_PIN2

LL_PWR_WAKEUP_PIN3

LL_PWR_WAKEUP_PIN4

LL_PWR_WAKEUP_PIN5

LL_PWR_WAKEUP_PIN6

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

56 LL RCC Generic Driver

56.1 RCC Firmware driver registers structures

56.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32c0xx_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***
- ***uint32_t HCLK_Frequency***
- ***uint32_t PCLK1_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

56.2 RCC Firmware driver API description

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

56.2.1 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_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 in stop mode is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR HSIKERON LL_RCC_HSI_IsEnabledInStopMode

LL_RCC_HSIKER_SetDivider

Function name

__STATIC_INLINE void LL_RCC_HSIKER_SetDivider (uint32_t Divider)

Function description

Set HSIKER divider.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_HSIKER_DIV_1
  - LL_RCC_HSIKER_DIV_2
  - LL_RCC_HSIKER_DIV_3
  - LL_RCC_HSIKER_DIV_4
  - LL_RCC_HSIKER_DIV_5
  - LL_RCC_HSIKER_DIV_6
  - LL_RCC_HSIKER_DIV_7
  - LL_RCC_HSIKER_DIV_8

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CR HSIKERDIV LL_RCC_HSIKER_SetDivider

LL_RCC_HSIKER_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_HSIKER_GetDivider (void)

Function description

Get HSIKER divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_HSIKER_DIV_1
  - LL_RCC_HSIKER_DIV_2
  - LL_RCC_HSIKER_DIV_3
  - LL_RCC_HSIKER_DIV_4
  - LL_RCC_HSIKER_DIV_5
  - LL_RCC_HSIKER_DIV_6
  - LL_RCC_HSIKER_DIV_7
  - LL_RCC_HSIKER_DIV_8

Reference Manual to LL API cross reference:

- CR HSIKERDIV LL_RCC_HSIKER_GetDivider

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_SetDivider

Function name

__STATIC_INLINE void LL_RCC_HSI_SetDivider (uint32_t Divider)

Function description

Set HSI divider.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8
  - LL_RCC_HSI_DIV_16
  - LL_RCC_HSI_DIV_32
  - LL_RCC_HSI_DIV_64
  - LL_RCC_HSI_DIV_128

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_HSI_SetDivider

LL_RCC_HSI_GetDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_HSI_GetDivider (void)

Function description

Get HSI divider.

Return values

- **can:** be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8
  - LL_RCC_HSI_DIV_16
  - LL_RCC_HSI_DIV_32
  - LL_RCC_HSI_DIV_64
  - LL_RCC_HSI_DIV_128

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_HSI_GetDivider

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 = 0x00 and Max_Data = 0xFF

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:

- ICSCR 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:

- ICSCR 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:

- ICSCR HSITRIM LL_RCC_HSI_GetCalibTrimming

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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 LSERDY LL_RCC_LSE_IsReady

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:

- CSR1 LSECSSD LL_RCC_LSE_IsCSSDetected

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:

- CSR2 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:

- CSR2 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:

- CSR2 LSIRDY LL_RCC_LSI_IsReady

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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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_HSI
  - LL_RCC_SYS_CLKSOURCE_HSE
  - LL_RCC_SYS_CLKSOURCE_HSIUSB48 (*)
  - LL_RCC_SYS_CLKSOURCE_LSI
  - LL_RCC_SYS_CLKSOURCE_LSE

Return values

- **None:**

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- CFGR 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_HSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSE
  - LL_RCC_SYS_CLKSOURCE_STATUS_HSIUSB48 (*)
  - LL_RCC_SYS_CLKSOURCE_STATUS_LSI
  - LL_RCC_SYS_CLKSOURCE_STATUS_LSE

Notes

- (*) peripheral not available on all devices

Reference Manual to LL API cross reference:

- CFGR SWS LL_RCC_GetSysClkSource

LL_RCC_SetSYSDivider

Function name

__STATIC_INLINE void LL_RCC_SetSYSDivider (uint32_t Divider)

Function description

Set SYS DIV.

Parameters

- **Divider:** This parameter can be one of the following values:
  - LL_RCC_SYSCLK_DIV_1
  - LL_RCC_SYSCLK_DIV_2
  - LL_RCC_SYSCLK_DIV_3
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_5
  - LL_RCC_SYSCLK_DIV_6
  - LL_RCC_SYSCLK_DIV_7
  - LL_RCC_SYSCLK_DIV_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR SYSDIV LL_RCC_SetSYSDivider

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_HCLK_DIV_1
  - LL_RCC_HCLK_DIV_2
  - LL_RCC_HCLK_DIV_4
  - LL_RCC_HCLK_DIV_8
  - LL_RCC_HCLK_DIV_16
  - LL_RCC_HCLK_DIV_64
  - LL_RCC_HCLK_DIV_128
  - LL_RCC_HCLK_DIV_256
  - LL_RCC_HCLK_DIV_512

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR HPRE LL_RCC_SetAHBPrescaler

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:

- CFGR PPRE LL_RCC_SetAPB1Prescaler

LL_RCC_SetHSIDiv

Function name

__STATIC_INLINE void LL_RCC_SetHSIDiv (uint32_t HSIDiv)

Function description

Set HSI48 division factor.

Parameters

- **HSIDiv:** This parameter can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8
  - LL_RCC_HSI_DIV_16
  - LL_RCC_HSI_DIV_32
  - LL_RCC_HSI_DIV_64
  - LL_RCC_HSI_DIV_128

Return values

- **None:**

Notes

- HSIDIV parameter is only applied to SYSCLK_Frequency when HSI is used as system clock source.

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_SetHSIDiv

LL_RCC_SetHSIKERDiv

Function name

__STATIC_INLINE void LL_RCC_SetHSIKERDiv (uint32_t HSIKERDiv)

Function description

Set HSIKER division factor.

Parameters

- **HSIKERDiv:** This parameter can be one of the following values:
  - LL_RCC_HSIKER_DIV_1
  - LL_RCC_HSIKER_DIV_2
  - LL_RCC_HSIKER_DIV_3
  - LL_RCC_HSIKER_DIV_4
  - LL_RCC_HSIKER_DIV_5
  - LL_RCC_HSIKER_DIV_6
  - LL_RCC_HSIKER_DIV_7
  - LL_RCC_HSIKER_DIV_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSIKERDIV LL_RCC_SetHSIKERDiv

LL_RCC_GetSYSDivider

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSYSDivider (void)

Function description

Get SYS divider.

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_3
  - LL_RCC_SYSCLK_DIV_4
  - LL_RCC_SYSCLK_DIV_5
  - LL_RCC_SYSCLK_DIV_6
  - LL_RCC_SYSCLK_DIV_7
  - LL_RCC_SYSCLK_DIV_8

Reference Manual to LL API cross reference:

- CR SYSDIV LL_RCC_GetSYSDivider

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_HCLK_DIV_1
  - LL_RCC_HCLK_DIV_2
  - LL_RCC_HCLK_DIV_4
  - LL_RCC_HCLK_DIV_8
  - LL_RCC_HCLK_DIV_16
  - LL_RCC_HCLK_DIV_64
  - LL_RCC_HCLK_DIV_128
  - LL_RCC_HCLK_DIV_256
  - LL_RCC_HCLK_DIV_512

Reference Manual to LL API cross reference:

- CFGR HPRE LL_RCC_GetAHBPrescaler

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:

- CFGR PPRE LL_RCC_GetAPB1Prescaler

LL_RCC_GetHSIDiv

Function name

__STATIC_INLINE uint32_t LL_RCC_GetHSIDiv (void)

Function description

Get HSI48 Division factor.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8
  - LL_RCC_HSI_DIV_16
  - LL_RCC_HSI_DIV_32
  - LL_RCC_HSI_DIV_64
  - LL_RCC_HSI_DIV_128

Notes

- HSIDIV parameter is only applied to SYSCLK_Frequency when HSI is used as system clock source.

Reference Manual to LL API cross reference:

- CR HSIDIV LL_RCC_GetHSIDiv

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_HSI
  - LL_RCC_MCO1SOURCE_HSIUSB48 (*)
  - LL_RCC_MCO1SOURCE_HSE
  - 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
  - LL_RCC_MCO1_DIV_32
  - LL_RCC_MCO1_DIV_64
  - LL_RCC_MCO1_DIV_128
  - LL_RCC_MCO1_DIV_256 (*)
  - LL_RCC_MCO1_DIV_512 (*)
  - LL_RCC_MCO1_DIV_1024 (*)

Return values

- **None:**

Notes

- (*) not available on all devices

Reference Manual to LL API cross reference:

- CFGR MCO1 LL_RCC_ConfigMCO
- CFGR MCO1PRE LL_RCC_ConfigMCO
- CFGR MCO2 LL_RCC_ConfigMCO
- CFGR MCO2PRE LL_RCC_ConfigMCO

LL_RCC_ConfigMCO2

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO2 (uint32_t MCOxSource, uint32_t MCOxPrescaler)

Function description

Configure MCO2.

Parameters

- **MCOxSource:** This parameter can be one of the following values:
  - LL_RCC_MCO2SOURCE_NOCLOCK
  - LL_RCC_MCO2SOURCE_SYSCLK
  - LL_RCC_MCO2SOURCE_HSI
  - LL_RCC_MCO2SOURCE_HSIUSB48 (*)
  - LL_RCC_MCO2SOURCE_HSE
  - LL_RCC_MCO2SOURCE_LSI
  - LL_RCC_MCO2SOURCE_LSE
- **MCOxPrescaler:** This parameter can be one of the following values:
  - LL_RCC_MCO2_DIV_1
  - LL_RCC_MCO2_DIV_2
  - LL_RCC_MCO2_DIV_4
  - LL_RCC_MCO2_DIV_8
  - LL_RCC_MCO2_DIV_16
  - LL_RCC_MCO2_DIV_32
  - LL_RCC_MCO2_DIV_64
  - LL_RCC_MCO2_DIV_128
  - LL_RCC_MCO2_DIV_256 (*)
  - LL_RCC_MCO2_DIV_512 (*)
  - LL_RCC_MCO2_DIV_1024 (*)

Return values

- **None:**

Notes

- feature not available in all devices.
- (*) not available on all devices

Reference Manual to LL API cross reference:

- CFGR MCO2SEL LL_RCC_ConfigMCO2
- CFGR MCO2PRE LL_RCC_ConfigMCO2

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_PCLK1
  - LL_RCC_USART1_CLKSOURCE_SYSCLK
  - LL_RCC_USART1_CLKSOURCE_HSIKER
  - LL_RCC_USART1_CLKSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR USARTxSEL LL_RCC_SetUSARTClockSource

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_HSIKER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR I2C1SEL LL_RCC_SetI2CClockSource

LL_RCC_SetADCClockSource

Function name

__STATIC_INLINE void LL_RCC_SetADCClockSource (uint32_t ADCxSource)

Function description

Configure ADC clock source.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE_SYSCLK
  - LL_RCC_ADC_CLKSOURCE_HSIKER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR ADCSEL LL_RCC_SetADCClockSource

LL_RCC_SetI2SClockSource

Function name

__STATIC_INLINE void LL_RCC_SetI2SClockSource (uint32_t I2SxSource)

Function description

Configure I2Sx clock source.

Parameters

- **I2SxSource:** This parameter can be one of the following values:
  - LL_RCC_I2S1_CLKSOURCE_SYSCLK
  - LL_RCC_I2S1_CLKSOURCE_PIN
  - LL_RCC_I2S1_CLKSOURCE_HSIKER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR I2S1SEL LL_RCC_SetI2SClockSource

LL_RCC_SetFDCANClockSource

Function name

__STATIC_INLINE void LL_RCC_SetFDCANClockSource (uint32_t FDCANxSource)

Function description

Configure FDCANx clock source.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN1_CLKSOURCE_PCLK1
  - LL_RCC_FDCAN1_CLKSOURCE_HSE
  - LL_RCC_FDCAN1_CLKSOURCE_HSIKER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCIPR FDCAN1SEL LL_RCC_SetFDCANClockSource

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

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_USART1_CLKSOURCE_PCLK1
  - LL_RCC_USART1_CLKSOURCE_SYSCLK
  - LL_RCC_USART1_CLKSOURCE_HSIKER
  - LL_RCC_USART1_CLKSOURCE_LSE

Reference Manual to LL API cross reference:

- CCIPR USART1SEL LL_RCC_GetUSARTClockSource

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

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_HSIKER

Reference Manual to LL API cross reference:

- CCIPR I2C1SEL LL_RCC_GetI2CClockSource

LL_RCC_GetADCClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetADCClockSource (uint32_t ADCx)

Function description

Get ADCx clock source.

Parameters

- **ADCx:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE_HSIKER
  - LL_RCC_ADC_CLKSOURCE_SYSCLK

Reference Manual to LL API cross reference:

- CCIPR ADCSEL LL_RCC_GetADCClockSource

LL_RCC_GetI2SClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetI2SClockSource (uint32_t I2Sx)

Function description

Get I2Sx clock source.

Parameters

- **I2Sx:** This parameter can be one of the following values:
  - LL_RCC_I2S1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_I2S1_CLKSOURCE_PIN
  - LL_RCC_I2S1_CLKSOURCE_SYSCLK
  - LL_RCC_I2S1_CLKSOURCE_HSIKER

Reference Manual to LL API cross reference:

- CCIPR I2S LL_RCC_GetI2SClockSource

LL_RCC_GetFDCANClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetFDCANClockSource (uint32_t FDCANx)

Function description

Get FDCANx clock source.

Parameters

- **FDCANx:** This parameter can be one of the following values:
  - LL_RCC_FDCAN1_CLKSOURCE

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_FDCAN1_CLKSOURCE_PCLK1
  - LL_RCC_FDCAN1_CLKSOURCE_HSE
  - LL_RCC_FDCAN1_CLKSOURCE_HSIKER

Reference Manual to LL API cross reference:

- CCIPR FDCAN LL_RCC_GetFDCANClockSource

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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 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:

- CSR1 RTCRST 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:

- CSR1 RTCRST LL_RCC_ReleaseBackupDomainReset

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_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_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_LSECSS

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LSECSS (void)

Function description

Clear LSE Clock security system interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CICR LSECSSC LL_RCC_ClearFlag_LSECSS

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_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_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_LSECSS

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LSECSS (void)

Function description

Check if LSE Clock security system interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CIFR LSECSSF LL_RCC_IsActiveFlag_LSECSS

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:

- CSR2 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:

- CSR2 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 Option byte reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR2 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:

- CSR2 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:

- CSR2 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:

- CSR2 WWDGRSTF LL_RCC_IsActiveFlag_WWDGRST

LL_RCC_IsActiveFlag_PWRRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PWRRST (void)

Function description

Check if RCC flag BOR or POR/PDR reset is set or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CSR2 PWRRSTF LL_RCC_IsActiveFlag_PWRRST

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:

- CSR2 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_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_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_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_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_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_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: HSI ON and used as system clock sourceHSE OFFAHB and APB1 prescaler set to 1.CSS, MCO OFFAll interrupts disabled
- This function does not 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 and APB1 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 and/or PCLK1 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

Return values

- **USART:** 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 (*)

Return values

- **I2C:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that HSI oscillator is not ready

Notes

- (*) peripheral not available on all devices

LL_RCC_GetADCClockFreq

Function name

uint32_t LL_RCC_GetADCClockFreq (uint32_t ADCxSource)

Function description

Return ADCx clock frequency.

Parameters

- **ADCxSource:** This parameter can be one of the following values:
  - LL_RCC_ADC_CLKSOURCE

Return values

- **ADC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (HSI) is not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetI2SClockFreq

Function name

uint32_t LL_RCC_GetI2SClockFreq (uint32_t I2SxSource)

Function description

Return I2Sx clock frequency.

Parameters

- **I2SxSource:** This parameter can be one of the following values:
  - LL_RCC_I2S1_CLKSOURCE

Return values

- **I2S:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

LL_RCC_GetRTCClockFreq

Function name

uint32_t LL_RCC_GetRTCClockFreq (void)

Function description

Return RTC clock frequency.

Return values

- **RTC:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillators (LSI, LSE or HSE) are not ready
  - LL_RCC_PERIPH_FREQUENCY_NA indicates that no clock source selected

LL_RCC_GetFDCANClockFreq

Function name

uint32_t LL_RCC_GetFDCANClockFreq (uint32_t FDCANxSource)

Function description

Return FDCANx clock frequency.

Parameters

- **FDCANxSource:** This parameter can be one of the following values:
  - LL_RCC_FDCAN1_CLKSOURCE

Return values

- **FDCAN:** clock frequency (in Hz)
  - LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator is not ready

56.3 RCC Firmware driver defines

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

56.3.1 RCC

RCC

Peripheral ADC get clock source

LL_RCC_ADC_CLKSOURCE

ADC Clock source selection

Peripheral ADC clock source selection

LL_RCC_ADC_CLKSOURCE_SYSCLK

SYSCLK used as ADC clock

LL_RCC_ADC_CLKSOURCE_HSIKER

HSIKER kernel used as ADC 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

Clear Flags Defines

LL_RCC_CICR_LSIRDYC

LSI Ready Interrupt Clear

LL_RCC_CICR_LSERDYC

LSE Ready Interrupt Clear

LL_RCC_CICR_HSIRDYC

HSI Ready Interrupt Clear

LL_RCC_CICR_HSERDYC

HSE Ready Interrupt Clear

LL_RCC_CICR_LSECSSC

LSE Clock Security System Interrupt Clear

LL_RCC_CICR_CSSC

Clock Security System Interrupt Clear

Peripheral FDCAN1 get clock source

LL_RCC_FDCAN1_CLKSOURCE

FDCAN1 Clock source selection

Peripheral FDCAN1 clock source selection

LL_RCC_FDCAN1_CLKSOURCE_PCLK1

PCLK1 clock used as FDCAN1 clock source

LL_RCC_FDCAN1_CLKSOURCE_HSIKER

HSIKER clock used as FDCAN1 clock source

LL_RCC_FDCAN1_CLKSOURCE_HSE

HSE clock used as FDCAN1 clock source

Get Flags Defines

LL_RCC_CIFR_LSIRDYF

LSI Ready Interrupt flag

LL_RCC_CIFR_LSERDYF

LSE Ready Interrupt flag

LL_RCC_CIFR_HSIRDYF

HSI Ready Interrupt flag

LL_RCC_CIFR_HSERDYF

HSE Ready Interrupt flag

LL_RCC_CIFR_LSECSSF

LSE Clock Security System Interrupt flag

LL_RCC_CIFR_CSSF

Clock Security System Interrupt flag

LL_RCC_CSR_LPWRRSTF

Low-Power reset flag

LL_RCC_CSR_OBLRSTF

OBL reset flag

LL_RCC_CSR_PINRSTF

PIN reset flag

LL_RCC_CSR_SFTRSTF

Software Reset flag

LL_RCC_CSR_IWDGRSTF

Independent Watchdog reset flag

LL_RCC_CSR_WWDGRSTF

Window watchdog reset flag

LL_RCC_CSR_PWRRSTF

BOR or POR/PDR reset flag

HSI Kernel division factor

LL_RCC_HSIKER_DIV_1

HSI kernel not divided

LL_RCC_HSIKER_DIV_2

HSI kernel divided by 2

LL_RCC_HSIKER_DIV_3

HSI kernel divided by 3

LL_RCC_HSIKER_DIV_4

HSI kernel divided by 4

LL_RCC_HSIKER_DIV_5

HSI kernel divided by 5

LL_RCC_HSIKER_DIV_6

HSI kernel divided by 6

LL_RCC_HSIKER_DIV_7

HSI kernel divided by 7

LL_RCC_HSIKER_DIV_8

HSI kernel divided by 8

HSI division factor

LL_RCC_HSI_DIV_1

HSI not divided

LL_RCC_HSI_DIV_2

HSI divided by 2

LL_RCC_HSI_DIV_4

HSI divided by 4

LL_RCC_HSI_DIV_8

HSI divided by 8

LL_RCC_HSI_DIV_16

HSI divided by 16

LL_RCC_HSI_DIV_32

HSI divided by 32

LL_RCC_HSI_DIV_64

HSI divided by 64

LL_RCC_HSI_DIV_128

HSI divided by 128

Peripheral I2C get clock source

LL_RCC_I2C1_CLKSOURCE

I2C1 Clock source selection

LL_RCC_I2C2_CLKSOURCE

I2C2 Clock source selection

Peripheral I2C 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_HSIKER

HSIKER 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_HSIKER

HSIKER clock used as I2C2 clock source

Peripheral I2S get clock source

LL_RCC_I2S1_CLKSOURCE

I2S1 Clock source selection

Peripheral I2S clock source selection

LL_RCC_I2S1_CLKSOURCE_SYSCLK

SYSCLK clock used as I2S1 clock source

LL_RCC_I2S1_CLKSOURCE_HSIKER

HSIKER clock used as I2S1 clock source

LL_RCC_I2S1_CLKSOURCE_PIN

External clock used as I2S1 clock source

IT Defines

LL_RCC_CIER_LSIRDYIE

LSI Ready Interrupt Enable

LL_RCC_CIER_LSERDYIE

LSE Ready Interrupt Enable

LL_RCC_CIER_HSIRDYIE

HSI Ready Interrupt Enable

LL_RCC_CIER_HSERDYIE

HSE Ready Interrupt Enable

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

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_HSI

HSI16 selection as MCO1 source

LL_RCC_MCO1SOURCE_HSE

HSE selection as MCO1 source

LL_RCC_MCO1SOURCE_LSI

LSI selection as MCO1 source

LL_RCC_MCO1SOURCE_LSE

LSE selection as MCO1 source

MCO1 prescaler

LL_RCC_MCO1_DIV_1

MCO1 not divided

LL_RCC_MCO1_DIV_2

MCO1 divided by 2

LL_RCC_MCO1_DIV_4

MCO1 divided by 4

LL_RCC_MCO1_DIV_8

MCO1 divided by 8

LL_RCC_MCO1_DIV_16

MCO1 divided by 16

LL_RCC_MCO1_DIV_32

MCO1 divided by 32

LL_RCC_MCO1_DIV_64

MCO1 divided by 64

LL_RCC_MCO1_DIV_128

MCO1 divided by 128

LL_RCC_MCO1_DIV_256

MCO1 divided by 256

LL_RCC_MCO1_DIV_512

MCO1 divided by 512

LL_RCC_MCO1_DIV_1024

MCO1 divided by 1024

MCO2 SOURCE selection

LL_RCC_MCO2SOURCE_NOCLOCK

MCO2 output disabled, no clock on MCO2

LL_RCC_MCO2SOURCE_SYSCLK

SYSCLK selection as MCO2 source

LL_RCC_MCO2SOURCE_HSI

HSI16 selection as MCO2 source

LL_RCC_MCO2SOURCE_HSE

HSE selection as MCO2 source

LL_RCC_MCO2SOURCE_LSI

LSI selection as MCO2 source

LL_RCC_MCO2SOURCE_LSE

LSE selection as MCO2 source

MCO2 prescaler

LL_RCC_MCO2_DIV_1

MCO2 not divided

LL_RCC_MCO2_DIV_2

MCO2 divided by 2

LL_RCC_MCO2_DIV_4

MCO2 divided by 4

LL_RCC_MCO2_DIV_8

MCO2 divided by 8

LL_RCC_MCO2_DIV_16

MCO2 divided by 16

LL_RCC_MCO2_DIV_32

MCO2 divided by 32

LL_RCC_MCO2_DIV_64

MCO2 divided by 64

LL_RCC_MCO2_DIV_128

MCO2 divided by 128

LL_RCC_MCO2_DIV_256

MCO1 divided by 256

LL_RCC_MCO2_DIV_512

MCO1 divided by 512

LL_RCC_MCO2_DIV_1024

MCO1 divided by 1024

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

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

AHB prescaler

LL_RCC_SYSCLK_DIV_1

SYSCLK not divided

LL_RCC_SYSCLK_DIV_2

SYSCLK divided by 2

LL_RCC_SYSCLK_DIV_3

SYSCLK divided by 3

LL_RCC_SYSCLK_DIV_4

SYSCLK divided by 4

LL_RCC_SYSCLK_DIV_5

SYSCLK divided by 5

LL_RCC_SYSCLK_DIV_6

SYSCLK divided by 6

LL_RCC_SYSCLK_DIV_7

SYSCLK divided by 7

LL_RCC_SYSCLK_DIV_8

SYSCLK divided by 8

System clock switch

LL_RCC_SYS_CLKSOURCE_HSI

HSI selection as system clock

LL_RCC_SYS_CLKSOURCE_HSE

HSE selection as system clock

LL_RCC_SYS_CLKSOURCE_LSI

LSI selection used as system clock

LL_RCC_SYS_CLKSOURCE_LSE

LSE selection used as system clock

System clock switch status

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_LSI

LSI used as system clock

LL_RCC_SYS_CLKSOURCE_STATUS_LSE

LSE used as system clock

Peripheral USARTx get clock source

LL_RCC_USART1_CLKSOURCE

USART1 Clock source selection

Peripheral USART clock source selection

LL_RCC_USART1_CLKSOURCE_PCLK1

PCLK1 clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_SYSCLK

SYSCLK clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_HSIKER

HSIKER clock used as USART1 clock source

LL_RCC_USART1_CLKSOURCE_LSE

LSE clock used as USART1 clock source

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

RCC Exported Macros

__LL_RCC_CALC_HCLK_FREQ

Description:

- Helper macro to calculate the HCLK frequency.

Parameters:

- __SYSCLKFREQ__: SYSCLK frequency (based on HSE/HSI/LSE/LSI)
- __AHBPRESCALER__: This parameter can be one of the following values:
  - LL_RCC_HCLK_DIV_1
  - LL_RCC_HCLK_DIV_2
  - LL_RCC_HCLK_DIV_4
  - LL_RCC_HCLK_DIV_8
  - LL_RCC_HCLK_DIV_16
  - LL_RCC_HCLK_DIV_64
  - LL_RCC_HCLK_DIV_128
  - LL_RCC_HCLK_DIV_256
  - LL_RCC_HCLK_DIV_512

Return value:

- HCLK: clock frequency (in Hz)

__LL_RCC_CALC_PCLK1_FREQ

Description:

- Helper macro to calculate the PCLK1 frequency (APB1)

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_HSI_FREQ

Description:

- Helper macro to calculate the HSISYS frequency.

Parameters:

- __HSIDIV__: This parameter can be one of the following values:
  - LL_RCC_HSI_DIV_1
  - LL_RCC_HSI_DIV_2
  - LL_RCC_HSI_DIV_4
  - LL_RCC_HSI_DIV_8
  - LL_RCC_HSI_DIV_16
  - LL_RCC_HSI_DIV_32
  - LL_RCC_HSI_DIV_64
  - LL_RCC_HSI_DIV_128

Return value:

- HSISYS: clock frequency (in Hz)

__LL_RCC_CALC_HSIKER_FREQ

Description:

- Helper macro to calculate the HSI Kernel frequency.

Parameters:

- __HSIKERDIV__: This parameter can be one of the following values:
  - LL_RCC_HSIKER_DIV_1
  - LL_RCC_HSIKER_DIV_2
  - LL_RCC_HSIKER_DIV_3
  - LL_RCC_HSIKER_DIV_4
  - LL_RCC_HSIKER_DIV_5
  - LL_RCC_HSIKER_DIV_6
  - LL_RCC_HSIKER_DIV_7
  - LL_RCC_HSIKER_DIV_8

Return value:

- HSIKER: clock frequency (in Hz)

57 LL RTC Generic Driver

57.1 RTC Firmware driver registers structures

57.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32c0xx_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()**.

57.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32c0xx_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 = 59This 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 = 59This feature can be modified afterwards using unitary function **LL_RTC_TIME_SetSecond()**.

57.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32c0xx_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 = 31This 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 = 99This feature can be modified afterwards using unitary function **LL_RTC_DATE_SetYear()**.

57.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32c0xx_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.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_SetMask()** for ALARM A.
- ***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 AThis feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()** for ALARM A
- ***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.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.

57.2 RTC Firmware driver API description

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

57.2.1 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

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

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_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 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_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

- **Sub:** second value (number between 0 and 65535)

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 (RTC_TypeDef * RTCx)

Function description

Get Alarm A 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:** Value between Min_Data=0x00 and Max_Data=0xF

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

- **Value:** between Min_Data=0x00 and Max_Data=0xF

Reference Manual to LL API cross reference:

- RTC_ALRMASSR MASKSS LL_RTC_ALMA_GetSubSecondMask

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:** Value between Min_Data=0x00 and Max_Data=0x7FFF

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

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- RCT_ALRMASSR SS LL_RTC_ALMA_GetSubSecond

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 ITSE 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 ITSE 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

- **Value:** between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- RTC_TSDR SS LL_RTC_TS_GetSubSecond

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

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

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

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_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_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_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_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_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_ALRAW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (const RTC_TypeDef * RTCx)

Function description

Get Alarm A write flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- RTC_ICSR ALRAWF LL_RTC_IsActiveFlag_ALRAW

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_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_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_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_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_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_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_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_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.

57.3 RTC Firmware driver defines

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

57.3.1 RTC

RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM is open-drain output

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM is push-pull 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

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

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 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_ITSF

LL_RTC_SCR_TSOVF

LL_RTC_SCR_TSF

LL_RTC_SCR_ALRAF

LL_RTC_ICSR_RECALPF

LL_RTC_ICSR_INITF

LL_RTC_ICSR_RSF

LL_RTC_ICSR_INITS

LL_RTC_ICSR_SHPF

LL_RTC_ICSR_ALRAWF

HOUR FORMAT

LL_RTC_HOURFORMAT_24HOUR

24 hour/day format

LL_RTC_HOURFORMAT_AMPM

AM/PM hour format

IT Defines

LL_RTC_CR_TSIE

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 is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW

Pin is low when ALRAF is asserted (depending on OSEL)

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

Delay (seconds) = SUBFS / (PREDIV_S + 1)

LL_RTC_SHIFT_SECOND_ADVANCE

Advance (seconds) = (1 - (SUBFS / (PREDIV_S + 1)))

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

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

58 LL SPI Generic Driver

58.1 SPI Firmware driver registers structures

58.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32c0xx_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 = 0xFFFF.This feature can be modified afterwards using unitary function **LL_SPI_SetCRCPolynomial()**.

58.1.2 LL_I2S_InitTypeDef

LL_I2S_InitTypeDef is defined in the stm32c0xx_ll_spi.h

Data Fields

- ***uint32_t Mode***
- ***uint32_t Standard***
- ***uint32_t DataFormat***
- ***uint32_t MCLKOutput***
- ***uint32_t AudioFreq***
- ***uint32_t ClockPolarity***

Field Documentation

- ***uint32_t LL_I2S_InitTypeDef::Mode***
  Specifies the I2S operating mode. This parameter can be a value of ***I2S_LL_EC_MODE***This feature can be modified afterwards using unitary function **LL_I2S_SetTransferMode()**.
- ***uint32_t LL_I2S_InitTypeDef::Standard***
  Specifies the standard used for the I2S communication. This parameter can be a value of ***I2S_LL_EC_STANDARD***This feature can be modified afterwards using unitary function **LL_I2S_SetStandard()**.
- ***uint32_t LL_I2S_InitTypeDef::DataFormat***
  Specifies the data format for the I2S communication. This parameter can be a value of ***I2S_LL_EC_DATA_FORMAT***This feature can be modified afterwards using unitary function **LL_I2S_SetDataFormat()**.
- ***uint32_t LL_I2S_InitTypeDef::MCLKOutput***
  Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of ***I2S_LL_EC_MCLK_OUTPUT***This feature can be modified afterwards using unitary functions **LL_I2S_EnableMasterClock()** or **LL_I2S_DisableMasterClock**.
- ***uint32_t LL_I2S_InitTypeDef::AudioFreq***
  Specifies the frequency selected for the I2S communication. This parameter can be a value of ***I2S_LL_EC_AUDIO_FREQ***Audio Frequency can be modified afterwards using Reference manual formulas to calculate Prescaler Linear, Parity and unitary functions **LL_I2S_SetPrescalerLinear()** and **LL_I2S_SetPrescalerParity()** to set it.
- ***uint32_t LL_I2S_InitTypeDef::ClockPolarity***
  Specifies the idle state of the I2S clock. This parameter can be a value of ***I2S_LL_EC_POLARITY***This feature can be modified afterwards using unitary function **LL_I2S_SetClockPolarity()**.

58.2 SPI Firmware driver API description

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

58.2.1 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_SetMode

Function name

__STATIC_INLINE void LL_SPI_SetMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set SPI operation 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 bit should not be changed when communication is ongoing.

Reference Manual to LL API cross reference:

- CR1 MSTR LL_SPI_SetMode
- CR1 SSI LL_SPI_SetMode

LL_SPI_GetMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetMode (const SPI_TypeDef * SPIx)

Function description

Get SPI operation 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:

- CR1 MSTR LL_SPI_GetMode
- CR1 SSI LL_SPI_GetMode

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 bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR2 FRF 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:

- CR2 FRF 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 bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 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:

- CR1 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 bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 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:

- CR1 CPOL LL_SPI_GetClockPolarity

LL_SPI_SetBaudRatePrescaler

Function name

__STATIC_INLINE void LL_SPI_SetBaudRatePrescaler (SPI_TypeDef * SPIx, uint32_t BaudRate)

Function description

Set baud rate prescaler.

Parameters

- **SPIx:** SPI Instance
- **BaudRate:** This parameter can be one of the following values:
  - 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

- These bits should not be changed when communication is ongoing. SPI BaudRate = fPCLK/Prescaler.

Reference Manual to LL API cross reference:

- CR1 BR LL_SPI_SetBaudRatePrescaler

LL_SPI_GetBaudRatePrescaler

Function name

__STATIC_INLINE uint32_t LL_SPI_GetBaudRatePrescaler (const SPI_TypeDef * SPIx)

Function description

Get baud rate prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - 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:

- CR1 BR 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 bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 LSBFIRST 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:

- CR1 LSBFIRST LL_SPI_GetTransferBitOrder

LL_SPI_SetTransferDirection

Function name

__STATIC_INLINE void LL_SPI_SetTransferDirection (SPI_TypeDef * SPIx, uint32_t TransferDirection)

Function description

Set transfer direction mode.

Parameters

- **SPIx:** SPI Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Return values

- **None:**

Notes

- For Half-Duplex mode, Rx Direction is set by default. 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 RXONLY LL_SPI_SetTransferDirection
- CR1 BIDIMODE LL_SPI_SetTransferDirection
- CR1 BIDIOE LL_SPI_SetTransferDirection

LL_SPI_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTransferDirection (const SPI_TypeDef * SPIx)

Function description

Get transfer direction mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_FULL_DUPLEX
  - LL_SPI_SIMPLEX_RX
  - LL_SPI_HALF_DUPLEX_RX
  - LL_SPI_HALF_DUPLEX_TX

Reference Manual to LL API cross reference:

- CR1 RXONLY LL_SPI_GetTransferDirection
- CR1 BIDIMODE LL_SPI_GetTransferDirection
- CR1 BIDIOE LL_SPI_GetTransferDirection

LL_SPI_SetDataWidth

Function name

__STATIC_INLINE void LL_SPI_SetDataWidth (SPI_TypeDef * SPIx, uint32_t DataWidth)

Function description

Set frame data width.

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

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DS LL_SPI_SetDataWidth

LL_SPI_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDataWidth (const SPI_TypeDef * SPIx)

Function description

Get frame data width.

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

Reference Manual to LL API cross reference:

- CR2 DS LL_SPI_GetDataWidth

LL_SPI_SetRxFIFOThreshold

Function name

__STATIC_INLINE void LL_SPI_SetRxFIFOThreshold (SPI_TypeDef * SPIx, uint32_t Threshold)

Function description

Set threshold of RXFIFO that triggers an RXNE event.

Parameters

- **SPIx:** SPI Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_SPI_RX_FIFO_TH_HALF
  - LL_SPI_RX_FIFO_TH_QUARTER

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 FRXTH LL_SPI_SetRxFIFOThreshold

LL_SPI_GetRxFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOThreshold (const SPI_TypeDef * SPIx)

Function description

Get threshold of RXFIFO that triggers an RXNE event.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_TH_HALF
  - LL_SPI_RX_FIFO_TH_QUARTER

Reference Manual to LL API cross reference:

- CR2 FRXTH LL_SPI_GetRxFIFOThreshold

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 bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 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:**

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 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).

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 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_8BIT
  - LL_SPI_CRC_16BIT

Return values

- **None:**

Notes

- This bit should be written only when SPI is disabled (SPE = 0) for correct operation.

Reference Manual to LL API cross reference:

- CR1 CRCL 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_8BIT
  - LL_SPI_CRC_16BIT

Reference Manual to LL API cross reference:

- CR1 CRCL LL_SPI_GetCRCWidth

LL_SPI_SetCRCNext

Function name

__STATIC_INLINE void LL_SPI_SetCRCNext (SPI_TypeDef * SPIx)

Function description

Set CRCNext to transfer CRC on the line.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit has to be written as soon as the last data is written in the SPIx_DR register.

Reference Manual to LL API cross reference:

- CR1 CRCNEXT LL_SPI_SetCRCNext

LL_SPI_SetCRCPolynomial

Function name

__STATIC_INLINE void LL_SPI_SetCRCPolynomial (SPI_TypeDef * SPIx, uint32_t CRCPoly)

Function description

Set polynomial for CRC calculation.

Parameters

- **SPIx:** SPI Instance
- **CRCPoly:** This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- CRCPR 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 calculation.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- CRCPR CRCPOLY LL_SPI_GetCRCPolynomial

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

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

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

- **Returned:** value is a number between Min_Data = 0x00 and Max_Data = 0xFFFF

Reference Manual to LL API cross reference:

- TXCRCR TXCRC LL_SPI_GetTxCRC

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

- LL_SPI_NSS_SOFT Mode is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR1 SSM LL_SPI_SetNSSMode
-
- CR2 SSOE LL_SPI_SetNSSMode

LL_SPI_GetNSSMode

Function name

__STATIC_INLINE uint32_t LL_SPI_GetNSSMode (const SPI_TypeDef * SPIx)

Function description

Get 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:

- CR1 SSM LL_SPI_GetNSSMode
-
- CR2 SSOE LL_SPI_GetNSSMode

LL_SPI_EnableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_EnableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Enable NSS pulse management.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR2 NSSP LL_SPI_EnableNSSPulseMgt

LL_SPI_DisableNSSPulseMgt

Function name

__STATIC_INLINE void LL_SPI_DisableNSSPulseMgt (SPI_TypeDef * SPIx)

Function description

Disable NSS pulse management.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR2 NSSP 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).

Notes

- This bit should not be changed when communication is ongoing. This bit is not used in SPI TI mode.

Reference Manual to LL API cross reference:

- CR2 NSSP LL_SPI_IsEnabledNSSPulse

LL_SPI_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_RXNE (const SPI_TypeDef * SPIx)

Function description

Check if Rx buffer is not empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_SPI_IsActiveFlag_RXNE

LL_SPI_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_TXE (const SPI_TypeDef * SPIx)

Function description

Check if Tx buffer is empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXE LL_SPI_IsActiveFlag_TXE

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 CRCERR 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_BSY

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_BSY (const SPI_TypeDef * SPIx)

Function description

Get busy flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- The BSY flag is cleared under any one of the following conditions: -When the SPI is correctly disabled -When a fault is detected in Master mode (MODF bit set to 1) -In Master mode, when it finishes a data transmission and no new data is ready to be sent -In Slave mode, when the BSY flag is set to '0' for at least one SPI clock cycle between each data transfer.

Reference Manual to LL API cross reference:

- SR BSY LL_SPI_IsActiveFlag_BSY

LL_SPI_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR FRE LL_SPI_IsActiveFlag_FRE

LL_SPI_GetRxFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetRxFIFOLevel (const SPI_TypeDef * SPIx)

Function description

Get FIFO reception Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_RX_FIFO_EMPTY
  - LL_SPI_RX_FIFO_QUARTER_FULL
  - LL_SPI_RX_FIFO_HALF_FULL
  - LL_SPI_RX_FIFO_FULL

Reference Manual to LL API cross reference:

- SR FRLVL LL_SPI_GetRxFIFOLevel

LL_SPI_GetTxFIFOLevel

Function name

__STATIC_INLINE uint32_t LL_SPI_GetTxFIFOLevel (const SPI_TypeDef * SPIx)

Function description

Get FIFO Transmission Level.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_TX_FIFO_EMPTY
  - LL_SPI_TX_FIFO_QUARTER_FULL
  - LL_SPI_TX_FIFO_HALF_FULL
  - LL_SPI_TX_FIFO_FULL

Reference Manual to LL API cross reference:

- SR FTLVL LL_SPI_GetTxFIFOLevel

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:

- SR CRCERR 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:**

Notes

- Clearing this flag is done by a read access to the SPIx_SR register followed by a write access to the SPIx_CR1 register

Reference Manual to LL API cross reference:

- SR MODF LL_SPI_ClearFlag_MODF

LL_SPI_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_OVR (const SPI_TypeDef * SPIx)

Function description

Clear overrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Clearing this flag is done by a read access to the SPIx_DR register followed by a read access to the SPIx_SR register

Reference Manual to LL API cross reference:

- SR OVR LL_SPI_ClearFlag_OVR

LL_SPI_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_SPI_ClearFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Clear frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- Clearing this flag is done by reading SPIx_SR register

Reference Manual to LL API cross reference:

- SR FRE LL_SPI_ClearFlag_FRE

LL_SPI_EnableIT_ERR

Function name

__STATIC_INLINE void LL_SPI_EnableIT_ERR (SPI_TypeDef * SPIx)

Function description

Enable error interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_SPI_EnableIT_ERR

LL_SPI_EnableIT_RXNE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Enable Rx buffer not empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_SPI_EnableIT_RXNE

LL_SPI_EnableIT_TXE

Function name

__STATIC_INLINE void LL_SPI_EnableIT_TXE (SPI_TypeDef * SPIx)

Function description

Enable Tx buffer empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_SPI_EnableIT_TXE

LL_SPI_DisableIT_ERR

Function name

__STATIC_INLINE void LL_SPI_DisableIT_ERR (SPI_TypeDef * SPIx)

Function description

Disable error interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (CRCERR, OVR, MODF in SPI mode, FRE at TI mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_SPI_DisableIT_ERR

LL_SPI_DisableIT_RXNE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Disable Rx buffer not empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_SPI_DisableIT_RXNE

LL_SPI_DisableIT_TXE

Function name

__STATIC_INLINE void LL_SPI_DisableIT_TXE (SPI_TypeDef * SPIx)

Function description

Disable Tx buffer empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_SPI_DisableIT_TXE

LL_SPI_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_ERR (const SPI_TypeDef * SPIx)

Function description

Check if error interrupt is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_SPI_IsEnabledIT_ERR

LL_SPI_IsEnabledIT_RXNE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_RXNE (const SPI_TypeDef * SPIx)

Function description

Check if Rx buffer not empty interrupt is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_SPI_IsEnabledIT_RXNE

LL_SPI_IsEnabledIT_TXE

Function name

__STATIC_INLINE uint32_t LL_SPI_IsEnabledIT_TXE (const SPI_TypeDef * SPIx)

Function description

Check if Tx buffer empty interrupt.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_SPI_IsEnabledIT_TXE

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:

- CR2 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:

- CR2 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:

- CR2 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:

- CR2 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:

- CR2 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:

- CR2 TXDMAEN LL_SPI_IsEnabledDMAReq_TX

LL_SPI_SetDMAParity_RX

Function name

__STATIC_INLINE void LL_SPI_SetDMAParity_RX (SPI_TypeDef * SPIx, uint32_t Parity)

Function description

Set parity of Last DMA reception.

Parameters

- **SPIx:** SPI Instance
- **Parity:** This parameter can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 LDMARX LL_SPI_SetDMAParity_RX

LL_SPI_GetDMAParity_RX

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_RX (const SPI_TypeDef * SPIx)

Function description

Get parity configuration for Last DMA reception.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Reference Manual to LL API cross reference:

- CR2 LDMARX LL_SPI_GetDMAParity_RX

LL_SPI_SetDMAParity_TX

Function name

__STATIC_INLINE void LL_SPI_SetDMAParity_TX (SPI_TypeDef * SPIx, uint32_t Parity)

Function description

Set parity of Last DMA transmission.

Parameters

- **SPIx:** SPI Instance
- **Parity:** This parameter can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 LDMATX LL_SPI_SetDMAParity_TX

LL_SPI_GetDMAParity_TX

Function name

__STATIC_INLINE uint32_t LL_SPI_GetDMAParity_TX (const SPI_TypeDef * SPIx)

Function description

Get parity configuration for Last DMA transmission.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_SPI_DMA_PARITY_ODD
  - LL_SPI_DMA_PARITY_EVEN

Reference Manual to LL API cross reference:

- CR2 LDMATX LL_SPI_GetDMAParity_TX

LL_SPI_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_SPI_DMA_GetRegAddr (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:

- DR DR LL_SPI_DMA_GetRegAddr

LL_SPI_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_SPI_ReceiveData8 (SPI_TypeDef * SPIx)

Function description

Read 8-Bits in the data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- DR DR LL_SPI_ReceiveData8

LL_SPI_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_SPI_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read 16-Bits in the data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- DR DR LL_SPI_ReceiveData16

LL_SPI_TransmitData8

Function name

__STATIC_INLINE void LL_SPI_TransmitData8 (SPI_TypeDef * SPIx, uint8_t TxData)

Function description

Write 8-Bits in the data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_SPI_TransmitData8

LL_SPI_TransmitData16

Function name

__STATIC_INLINE void LL_SPI_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write 16-Bits in the data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_SPI_TransmitData16

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 peripheral 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:**

LL_I2S_Enable

Function name

__STATIC_INLINE void LL_I2S_Enable (SPI_TypeDef * SPIx)

Function description

Select I2S mode and Enable I2S peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SMOD LL_I2S_Enable
- I2SCFGR I2SE LL_I2S_Enable

LL_I2S_Disable

Function name

__STATIC_INLINE void LL_I2S_Disable (SPI_TypeDef * SPIx)

Function description

Disable I2S peripheral.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SE LL_I2S_Disable

LL_I2S_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabled (const SPI_TypeDef * SPIx)

Function description

Check if I2S peripheral is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SCFGR I2SE LL_I2S_IsEnabled

LL_I2S_SetDataFormat

Function name

__STATIC_INLINE void LL_I2S_SetDataFormat (SPI_TypeDef * SPIx, uint32_t DataFormat)

Function description

Set I2S data frame length.

Parameters

- **SPIx:** SPI Instance
- **DataFormat:** This parameter can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_SetDataFormat
- I2SCFGR CHLEN LL_I2S_SetDataFormat

LL_I2S_GetDataFormat

Function name

__STATIC_INLINE uint32_t LL_I2S_GetDataFormat (const SPI_TypeDef * SPIx)

Function description

Get I2S data frame length.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_DATAFORMAT_16B
  - LL_I2S_DATAFORMAT_16B_EXTENDED
  - LL_I2S_DATAFORMAT_24B
  - LL_I2S_DATAFORMAT_32B

Reference Manual to LL API cross reference:

- I2SCFGR DATLEN LL_I2S_GetDataFormat
- I2SCFGR CHLEN LL_I2S_GetDataFormat

LL_I2S_SetClockPolarity

Function name

__STATIC_INLINE void LL_I2S_SetClockPolarity (SPI_TypeDef * SPIx, uint32_t ClockPolarity)

Function description

Set I2S clock polarity.

Parameters

- **SPIx:** SPI Instance
- **ClockPolarity:** This parameter can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_SetClockPolarity

LL_I2S_GetClockPolarity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetClockPolarity (const SPI_TypeDef * SPIx)

Function description

Get I2S clock polarity.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_POLARITY_LOW
  - LL_I2S_POLARITY_HIGH

Reference Manual to LL API cross reference:

- I2SCFGR CKPOL LL_I2S_GetClockPolarity

LL_I2S_SetStandard

Function name

__STATIC_INLINE void LL_I2S_SetStandard (SPI_TypeDef * SPIx, uint32_t Standard)

Function description

Set I2S standard protocol.

Parameters

- **SPIx:** SPI Instance
- **Standard:** This parameter can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_SetStandard
- I2SCFGR PCMSYNC LL_I2S_SetStandard

LL_I2S_GetStandard

Function name

__STATIC_INLINE uint32_t LL_I2S_GetStandard (const SPI_TypeDef * SPIx)

Function description

Get I2S standard protocol.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_STANDARD_PHILIPS
  - LL_I2S_STANDARD_MSB
  - LL_I2S_STANDARD_LSB
  - LL_I2S_STANDARD_PCM_SHORT
  - LL_I2S_STANDARD_PCM_LONG

Reference Manual to LL API cross reference:

- I2SCFGR I2SSTD LL_I2S_GetStandard
- I2SCFGR PCMSYNC LL_I2S_GetStandard

LL_I2S_SetTransferMode

Function name

__STATIC_INLINE void LL_I2S_SetTransferMode (SPI_TypeDef * SPIx, uint32_t Mode)

Function description

Set I2S transfer mode.

Parameters

- **SPIx:** SPI Instance
- **Mode:** This parameter can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_SetTransferMode

LL_I2S_GetTransferMode

Function name

__STATIC_INLINE uint32_t LL_I2S_GetTransferMode (const SPI_TypeDef * SPIx)

Function description

Get I2S transfer mode.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_MODE_SLAVE_TX
  - LL_I2S_MODE_SLAVE_RX
  - LL_I2S_MODE_MASTER_TX
  - LL_I2S_MODE_MASTER_RX

Reference Manual to LL API cross reference:

- I2SCFGR I2SCFG LL_I2S_GetTransferMode

LL_I2S_SetPrescalerLinear

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerLinear (SPI_TypeDef * SPIx, uint8_t PrescalerLinear)

Function description

Set I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR I2SDIV LL_I2S_SetPrescalerLinear

LL_I2S_GetPrescalerLinear

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerLinear (const SPI_TypeDef * SPIx)

Function description

Get I2S linear prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **PrescalerLinear:** Value between Min_Data=0x02 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- I2SPR I2SDIV LL_I2S_GetPrescalerLinear

LL_I2S_SetPrescalerParity

Function name

__STATIC_INLINE void LL_I2S_SetPrescalerParity (SPI_TypeDef * SPIx, uint32_t PrescalerParity)

Function description

Set I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR ODD LL_I2S_SetPrescalerParity

LL_I2S_GetPrescalerParity

Function name

__STATIC_INLINE uint32_t LL_I2S_GetPrescalerParity (const SPI_TypeDef * SPIx)

Function description

Get I2S parity prescaler.

Parameters

- **SPIx:** SPI Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Reference Manual to LL API cross reference:

- I2SPR ODD LL_I2S_GetPrescalerParity

LL_I2S_EnableMasterClock

Function name

__STATIC_INLINE void LL_I2S_EnableMasterClock (SPI_TypeDef * SPIx)

Function description

Enable the master clock output (Pin MCK)

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_EnableMasterClock

LL_I2S_DisableMasterClock

Function name

__STATIC_INLINE void LL_I2S_DisableMasterClock (SPI_TypeDef * SPIx)

Function description

Disable the master clock output (Pin MCK)

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_DisableMasterClock

LL_I2S_IsEnabledMasterClock

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledMasterClock (const SPI_TypeDef * SPIx)

Function description

Check if the master clock output (Pin MCK) is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SPR MCKOE LL_I2S_IsEnabledMasterClock

LL_I2S_EnableAsyncStart

Function name

__STATIC_INLINE void LL_I2S_EnableAsyncStart (SPI_TypeDef * SPIx)

Function description

Enable asynchronous start.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_EnableAsyncStart

LL_I2S_DisableAsyncStart

Function name

__STATIC_INLINE void LL_I2S_DisableAsyncStart (SPI_TypeDef * SPIx)

Function description

Disable asynchronous start.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_DisableAsyncStart

LL_I2S_IsEnabledAsyncStart

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledAsyncStart (const SPI_TypeDef * SPIx)

Function description

Check if asynchronous start is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- I2SCFGR ASTRTEN LL_I2S_IsEnabledAsyncStart

LL_I2S_IsActiveFlag_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_RXNE (const SPI_TypeDef * SPIx)

Function description

Check if Rx buffer is not empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR RXNE LL_I2S_IsActiveFlag_RXNE

LL_I2S_IsActiveFlag_TXE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_TXE (const SPI_TypeDef * SPIx)

Function description

Check if Tx buffer is empty.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR TXE LL_I2S_IsActiveFlag_TXE

LL_I2S_IsActiveFlag_BSY

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_BSY (const SPI_TypeDef * SPIx)

Function description

Get busy flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR BSY LL_I2S_IsActiveFlag_BSY

LL_I2S_IsActiveFlag_OVR

Function name

__STATIC_INLINE uint32_t LL_I2S_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_I2S_IsActiveFlag_OVR

LL_I2S_IsActiveFlag_UDR

Function name

__STATIC_INLINE uint32_t LL_I2S_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_I2S_IsActiveFlag_UDR

LL_I2S_IsActiveFlag_FRE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Get frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SR FRE LL_I2S_IsActiveFlag_FRE

LL_I2S_IsActiveFlag_CHSIDE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsActiveFlag_CHSIDE (const SPI_TypeDef * SPIx)

Function description

Get channel side flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Notes

- 0: Channel Left has to be transmitted or has been received 1: Channel Right has to be transmitted or has been received It has no significance in PCM mode.

Reference Manual to LL API cross reference:

- SR CHSIDE LL_I2S_IsActiveFlag_CHSIDE

LL_I2S_ClearFlag_OVR

Function name

__STATIC_INLINE void LL_I2S_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:

- SR OVR LL_I2S_ClearFlag_OVR

LL_I2S_ClearFlag_UDR

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_UDR (const SPI_TypeDef * SPIx)

Function description

Clear underrun error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR UDR LL_I2S_ClearFlag_UDR

LL_I2S_ClearFlag_FRE

Function name

__STATIC_INLINE void LL_I2S_ClearFlag_FRE (const SPI_TypeDef * SPIx)

Function description

Clear frame format error flag.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR FRE LL_I2S_ClearFlag_FRE

LL_I2S_EnableIT_ERR

Function name

__STATIC_INLINE void LL_I2S_EnableIT_ERR (SPI_TypeDef * SPIx)

Function description

Enable error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_EnableIT_ERR

LL_I2S_EnableIT_RXNE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Enable Rx buffer not empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_EnableIT_RXNE

LL_I2S_EnableIT_TXE

Function name

__STATIC_INLINE void LL_I2S_EnableIT_TXE (SPI_TypeDef * SPIx)

Function description

Enable Tx buffer empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_EnableIT_TXE

LL_I2S_DisableIT_ERR

Function name

__STATIC_INLINE void LL_I2S_DisableIT_ERR (SPI_TypeDef * SPIx)

Function description

Disable error IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Notes

- This bit controls the generation of an interrupt when an error condition occurs (OVR, UDR and FRE in I2S mode).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_DisableIT_ERR

LL_I2S_DisableIT_RXNE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_RXNE (SPI_TypeDef * SPIx)

Function description

Disable Rx buffer not empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_DisableIT_RXNE

LL_I2S_DisableIT_TXE

Function name

__STATIC_INLINE void LL_I2S_DisableIT_TXE (SPI_TypeDef * SPIx)

Function description

Disable Tx buffer empty IT.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_DisableIT_TXE

LL_I2S_IsEnabledIT_ERR

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_ERR (const SPI_TypeDef * SPIx)

Function description

Check if ERR IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 ERRIE LL_I2S_IsEnabledIT_ERR

LL_I2S_IsEnabledIT_RXNE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_RXNE (const SPI_TypeDef * SPIx)

Function description

Check if RXNE IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 RXNEIE LL_I2S_IsEnabledIT_RXNE

LL_I2S_IsEnabledIT_TXE

Function name

__STATIC_INLINE uint32_t LL_I2S_IsEnabledIT_TXE (const SPI_TypeDef * SPIx)

Function description

Check if TXE IT is enabled.

Parameters

- **SPIx:** SPI Instance

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR2 TXEIE LL_I2S_IsEnabledIT_TXE

LL_I2S_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Enable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_EnableDMAReq_RX

LL_I2S_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_RX (SPI_TypeDef * SPIx)

Function description

Disable DMA Rx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXDMAEN LL_I2S_DisableDMAReq_RX

LL_I2S_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_I2S_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:

- CR2 RXDMAEN LL_I2S_IsEnabledDMAReq_RX

LL_I2S_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_EnableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Enable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_EnableDMAReq_TX

LL_I2S_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_I2S_DisableDMAReq_TX (SPI_TypeDef * SPIx)

Function description

Disable DMA Tx.

Parameters

- **SPIx:** SPI Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXDMAEN LL_I2S_DisableDMAReq_TX

LL_I2S_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_I2S_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:

- CR2 TXDMAEN LL_I2S_IsEnabledDMAReq_TX

LL_I2S_ReceiveData16

Function name

__STATIC_INLINE uint16_t LL_I2S_ReceiveData16 (SPI_TypeDef * SPIx)

Function description

Read 16-Bits in data register.

Parameters

- **SPIx:** SPI Instance

Return values

- **RxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Reference Manual to LL API cross reference:

- DR DR LL_I2S_ReceiveData16

LL_I2S_TransmitData16

Function name

__STATIC_INLINE void LL_I2S_TransmitData16 (SPI_TypeDef * SPIx, uint16_t TxData)

Function description

Write 16-Bits in data register.

Parameters

- **SPIx:** SPI Instance
- **TxData:** Value between Min_Data=0x0000 and Max_Data=0xFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- DR DR LL_I2S_TransmitData16

LL_I2S_DeInit

Function name

ErrorStatus LL_I2S_DeInit (const SPI_TypeDef * SPIx)

Function description

De-initialize the SPI/I2S 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_I2S_Init

Function name

ErrorStatus LL_I2S_Init (SPI_TypeDef * SPIx, LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Initializes the SPI/I2S registers according to the specified parameters in I2S_InitStruct.

Parameters

- **SPIx:** SPI Instance
- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: SPI registers are Initialized
  - ERROR: SPI registers are not Initialized

Notes

- As some bits in SPI configuration registers can only be written when the SPI is disabled (SPI_CR1_SPE bit =0), SPI peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.

LL_I2S_StructInit

Function name

void LL_I2S_StructInit (LL_I2S_InitTypeDef * I2S_InitStruct)

Function description

Set each LL_I2S_InitTypeDef field to default value.

Parameters

- **I2S_InitStruct:** pointer to a LL_I2S_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

LL_I2S_ConfigPrescaler

Function name

void LL_I2S_ConfigPrescaler (SPI_TypeDef * SPIx, uint32_t PrescalerLinear, uint32_t PrescalerParity)

Function description

Set linear and parity prescaler.

Parameters

- **SPIx:** SPI Instance
- **PrescalerLinear:** value Min_Data=0x02 and Max_Data=0xFF.
- **PrescalerParity:** This parameter can be one of the following values:
  - LL_I2S_PRESCALER_PARITY_EVEN
  - LL_I2S_PRESCALER_PARITY_ODD

Return values

- **None:**

Notes

- To calculate value of PrescalerLinear(I2SDIV[7:0] bits) and PrescalerParity(ODD bit) Check Audio frequency table and formulas inside Reference Manual (SPI/I2S).

58.3 SPI Firmware driver defines

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

58.3.1 SPI

SPI

Baud Rate Prescaler

LL_SPI_BAUDRATEPRESCALER_DIV2

BaudRate control equal to fPCLK/2

LL_SPI_BAUDRATEPRESCALER_DIV4

BaudRate control equal to fPCLK/4

LL_SPI_BAUDRATEPRESCALER_DIV8

BaudRate control equal to fPCLK/8

LL_SPI_BAUDRATEPRESCALER_DIV16

BaudRate control equal to fPCLK/16

LL_SPI_BAUDRATEPRESCALER_DIV32

BaudRate control equal to fPCLK/32

LL_SPI_BAUDRATEPRESCALER_DIV64

BaudRate control equal to fPCLK/64

LL_SPI_BAUDRATEPRESCALER_DIV128

BaudRate control equal to fPCLK/128

LL_SPI_BAUDRATEPRESCALER_DIV256

BaudRate control equal to fPCLK/256

Transmission Bit Order

LL_SPI_LSB_FIRST

Data is transmitted/received with the LSB first

LL_SPI_MSB_FIRST

Data is transmitted/received with the MSB first

CRC Calculation

LL_SPI_CRCCALCULATION_DISABLE

CRC calculation disabled

LL_SPI_CRCCALCULATION_ENABLE

CRC calculation enabled

CRC Length

LL_SPI_CRC_8BIT

8-bit CRC length

LL_SPI_CRC_16BIT

16-bit CRC length

Datawidth

LL_SPI_DATAWIDTH_4BIT

Data length for SPI transfer: 4 bits

LL_SPI_DATAWIDTH_5BIT

Data length for SPI transfer: 5 bits

LL_SPI_DATAWIDTH_6BIT

Data length for SPI transfer: 6 bits

LL_SPI_DATAWIDTH_7BIT

Data length for SPI transfer: 7 bits

LL_SPI_DATAWIDTH_8BIT

Data length for SPI transfer: 8 bits

LL_SPI_DATAWIDTH_9BIT

Data length for SPI transfer: 9 bits

LL_SPI_DATAWIDTH_10BIT

Data length for SPI transfer: 10 bits

LL_SPI_DATAWIDTH_11BIT

Data length for SPI transfer: 11 bits

LL_SPI_DATAWIDTH_12BIT

Data length for SPI transfer: 12 bits

LL_SPI_DATAWIDTH_13BIT

Data length for SPI transfer: 13 bits

LL_SPI_DATAWIDTH_14BIT

Data length for SPI transfer: 14 bits

LL_SPI_DATAWIDTH_15BIT

Data length for SPI transfer: 15 bits

LL_SPI_DATAWIDTH_16BIT

Data length for SPI transfer: 16 bits

DMA Parity

LL_SPI_DMA_PARITY_EVEN

Select DMA parity Even

LL_SPI_DMA_PARITY_ODD

Select DMA parity Odd

Get Flags Defines

LL_SPI_SR_RXNE

Rx buffer not empty flag

LL_SPI_SR_TXE

Tx buffer empty flag

LL_SPI_SR_BSY

Busy flag

LL_SPI_SR_CRCERR

CRC error flag

LL_SPI_SR_MODF

Mode fault flag

LL_SPI_SR_OVR

Overrun flag

LL_SPI_SR_FRE

TI mode frame format error flag

IT Defines

LL_SPI_CR2_RXNEIE

Rx buffer not empty interrupt enable

LL_SPI_CR2_TXEIE

Tx buffer empty interrupt enable

LL_SPI_CR2_ERRIE

Error interrupt enable

LL_I2S_CR2_RXNEIE

Rx buffer not empty interrupt enable

LL_I2S_CR2_TXEIE

Tx buffer empty interrupt enable

LL_I2S_CR2_ERRIE

Error interrupt enable

Operation Mode

LL_SPI_MODE_MASTER

Master configuration

LL_SPI_MODE_SLAVE

Slave configuration

Slave Select Pin Mode

LL_SPI_NSS_SOFT

NSS managed internally. NSS pin not used and free

LL_SPI_NSS_HARD_INPUT

NSS pin used in Input. Only used in Master mode

LL_SPI_NSS_HARD_OUTPUT

NSS pin used in Output. Only used in Slave mode as chip select

Clock Phase

LL_SPI_PHASE_1EDGE

First clock transition is the first data capture edge

LL_SPI_PHASE_2EDGE

Second clock transition is the first data capture edge

Clock Polarity

LL_SPI_POLARITY_LOW

Clock to 0 when idle

LL_SPI_POLARITY_HIGH

Clock to 1 when idle

Serial Protocol

LL_SPI_PROTOCOL_MOTOROLA

Motorola mode. Used as default value

LL_SPI_PROTOCOL_TI

TI mode

RX FIFO Level

LL_SPI_RX_FIFO_EMPTY

FIFO reception empty

LL_SPI_RX_FIFO_QUARTER_FULL

FIFO reception 1/4

LL_SPI_RX_FIFO_HALF_FULL

FIFO reception 1/2

LL_SPI_RX_FIFO_FULL

FIFO reception full

RX FIFO Threshold

LL_SPI_RX_FIFO_TH_HALF

RXNE event is generated if FIFO level is greater than or equal to 1/2 (16-bit)

LL_SPI_RX_FIFO_TH_QUARTER

RXNE event is generated if FIFO level is greater than or equal to 1/4 (8-bit)

Transfer Mode

LL_SPI_FULL_DUPLEX

Full-Duplex mode. Rx and Tx transfer on 2 lines

LL_SPI_SIMPLEX_RX

Simplex Rx mode. Rx transfer only on 1 line

LL_SPI_HALF_DUPLEX_RX

Half-Duplex Rx mode. Rx transfer on 1 line

LL_SPI_HALF_DUPLEX_TX

Half-Duplex Tx mode. Tx transfer on 1 line

TX FIFO Level

LL_SPI_TX_FIFO_EMPTY

FIFO transmission empty

LL_SPI_TX_FIFO_QUARTER_FULL

FIFO transmission 1/4

LL_SPI_TX_FIFO_HALF_FULL

FIFO transmission 1/2

LL_SPI_TX_FIFO_FULL

FIFO transmission full

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

59 LL SYSTEM Generic Driver

59.1 SYSTEM Firmware driver API description

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

59.1.1 Detailed description of functions

LL_SYSCFG_SetRemapMemory

Function name

__STATIC_INLINE void LL_SYSCFG_SetRemapMemory (uint32_t Memory)

Function description

Set memory mapping at address 0x00000000.

Parameters

- **Memory:** This parameter can be one of the following values:
  - LL_SYSCFG_REMAP_FLASH
  - LL_SYSCFG_REMAP_SYSTEMFLASH
  - LL_SYSCFG_REMAP_SRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 MEM_MODE LL_SYSCFG_SetRemapMemory

LL_SYSCFG_GetRemapMemory

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetRemapMemory (void)

Function description

Get memory mapping at address 0x00000000.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_REMAP_FLASH
  - LL_SYSCFG_REMAP_SYSTEMFLASH
  - LL_SYSCFG_REMAP_SRAM

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 MEM_MODE LL_SYSCFG_GetRemapMemory

LL_SYSCFG_EnablePinRemap

Function name

__STATIC_INLINE void LL_SYSCFG_EnablePinRemap (uint32_t PinRemap)

Function description

Enable remap of a pin on different pad.

Parameters

- **PinRemap:** This parameter can be a combination of the following values:
  - LL_SYSCFG_PIN_RMP_PA11
  - LL_SYSCFG_PIN_RMP_PA12

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 PA11_RMP LL_SYSCFG_EnablePinRemap
- SYSCFG_CFGR1 PA12_RMP LL_SYSCFG_EnablePinRemap
-

LL_SYSCFG_DisablePinRemap

Function name

__STATIC_INLINE void LL_SYSCFG_DisablePinRemap (uint32_t PinRemap)

Function description

Enable remap of a pin on different pad.

Parameters

- **PinRemap:** This parameter can be a combination of the following values:
  - LL_SYSCFG_PIN_RMP_PA11
  - LL_SYSCFG_PIN_RMP_PA12

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 PA11_RMP LL_SYSCFG_DisablePinRemap
- SYSCFG_CFGR1 PA12_RMP LL_SYSCFG_DisablePinRemap
-

LL_SYSCFG_SetIRModEnvelopeSignal

Function name

__STATIC_INLINE void LL_SYSCFG_SetIRModEnvelopeSignal (uint32_t Source)

Function description

Set IR Modulation Envelope signal source.

Parameters

- **Source:** This parameter can be one of the following values:
  - LL_SYSCFG_IR_MOD_TIM16
  - LL_SYSCFG_IR_MOD_USART1
  - LL_SYSCFG_IR_MOD_USART2

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 IR_MOD LL_SYSCFG_SetIRModEnvelopeSignal

LL_SYSCFG_GetIRModEnvelopeSignal

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetIRModEnvelopeSignal (void)

Function description

Get IR Modulation Envelope signal source.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_IR_MOD_TIM16
  - LL_SYSCFG_IR_MOD_USART1
  - LL_SYSCFG_IR_MOD_USART2

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 IR_MOD LL_SYSCFG_GetIRModEnvelopeSignal

LL_SYSCFG_SetIRPolarity

Function name

__STATIC_INLINE void LL_SYSCFG_SetIRPolarity (uint32_t Polarity)

Function description

Set IR Output polarity.

Parameters

- **Polarity:** This parameter can be one of the following values:
  - LL_SYSCFG_IR_POL_INVERTED
  - LL_SYSCFG_IR_POL_NOT_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 IR_POL LL_SYSCFG_SetIRPolarity

LL_SYSCFG_GetIRPolarity

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetIRPolarity (void)

Function description

Get IR Output polarity.

Return values

- **Returned:** value can be one of the following values:
  - LL_SYSCFG_IR_POL_INVERTED
  - LL_SYSCFG_IR_POL_NOT_INVERTED

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 IR_POL LL_SYSCFG_GetIRPolarity

LL_SYSCFG_EnableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_EnableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Enable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA9
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA10
  - LL_SYSCFG_I2C_FASTMODEPLUS_PC14

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 I2C_FMP_PB6 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB7 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB8 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB9 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_I2C1 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_I2C2 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PA9 LL_SYSCFG_EnableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PA10 LL_SYSCFG_EnableFastModePlus SYSCFG_CFGR1 I2C_FMP_PC14 LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_DisableFastModePlus

Function name

__STATIC_INLINE void LL_SYSCFG_DisableFastModePlus (uint32_t ConfigFastModePlus)

Function description

Disable the I2C fast mode plus driving capability.

Parameters

- **ConfigFastModePlus:** This parameter can be a combination of the following values:
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB8
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB9
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C1
  - LL_SYSCFG_I2C_FASTMODEPLUS_I2C2 (*)
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA9
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA10
  - LL_SYSCFG_I2C_FASTMODEPLUS_PC14

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR1 I2C_FMP_PB6 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB7 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB8 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PB9 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_I2C1 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_I2C2 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PA9 LL_SYSCFG_DisableFastModePlus
- SYSCFG_CFGR1 I2C_FMP_PA10 LL_SYSCFG_DisableFastModePlus SYSCFG_CFGR1 I2C_FMP_PC14 LL_SYSCFG_EnableFastModePlus

LL_SYSCFG_SetTIMBreakInputs

Function name

__STATIC_INLINE void LL_SYSCFG_SetTIMBreakInputs (uint32_t Break)

Function description

Set connections to TIM1/16/17 Break inputs.

Parameters

- **Break:** This parameter can be a combination of the following values:
  - LL_SYSCFG_TIMBREAK_LOCKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 CLL LL_SYSCFG_SetTIMBreakInputs
-

LL_SYSCFG_GetTIMBreakInputs

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetTIMBreakInputs (void)

Function description

Get connections to TIM1/16/17 Break inputs.

Return values

- **Returned:** value can be can be a combination of the following values:
  - LL_SYSCFG_TIMBREAK_LOCKUP

Reference Manual to LL API cross reference:

- SYSCFG_CFGR2 CLL LL_SYSCFG_GetTIMBreakInputs
-

LL_SYSCFG_ConfigPinMux

Function name

__STATIC_INLINE void LL_SYSCFG_ConfigPinMux (uint32_t mux_cfg)

Function description

Config PinMux.

Parameters

- **mux_cfg:** This parameter can be a value of PINMUX Config

Return values

- **None:**

Reference Manual to LL API cross reference:

- SYSCFG_CFGR3 CLL LL_SYSCFG_ConfigPinMux
-

LL_SYSCFG_GetConfigPinMux

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetConfigPinMux (uint32_t LL_PINMUX_PACKx_PINy)

Function description

Get PinMux configuration.

Parameters

- **LL_PINMUX_PACKx_PINy:** This parameter can be a value of PINMUX Config Source

Return values

- **Returned:** value can be one of SYSTEM_LL_PINMUX_CFG defines

Reference Manual to LL API cross reference:

- SYSCFG_CFGR3 CLL LL_SYSCFG_GetConfigPinMux
-

LL_SYSCFG_IsActiveFlag_WWDG

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_WWDG (void)

Function description

Check if Window watchdog interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE0 SR_EWDG LL_SYSCFG_IsActiveFlag_WWDG

LL_SYSCFG_IsActiveFlag_RTC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_RTC (void)

Function description

Check if RTC interrupt occurred or not (EXTI line 19).

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE2 SR_RTC LL_SYSCFG_IsActiveFlag_RTC

LL_SYSCFG_IsActiveFlag_FLASH_ITF

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_FLASH_ITF (void)

Function description

Check if Flash interface interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE3 SR_FLASH_ITF LL_SYSCFG_IsActiveFlag_FLASH_ITF

LL_SYSCFG_IsActiveFlag_CLK_CTRL

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_CLK_CTRL (void)

Function description

Check if Reset and clock control interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE4_SR_RCC LL_SYSCFG_IsActiveFlag_CLK_CTRL

LL_SYSCFG_IsActiveFlag_EXTI0

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI0 (void)

Function description

Check if EXTI line 0 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE5 SR_EXTI0 LL_SYSCFG_IsActiveFlag_EXTI0

LL_SYSCFG_IsActiveFlag_EXTI1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI1 (void)

Function description

Check if EXTI line 1 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE5 SR_EXTI1 LL_SYSCFG_IsActiveFlag_EXTI1

LL_SYSCFG_IsActiveFlag_EXTI2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI2 (void)

Function description

Check if EXTI line 2 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE6 SR_EXTI2 LL_SYSCFG_IsActiveFlag_EXTI2

LL_SYSCFG_IsActiveFlag_EXTI3

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI3 (void)

Function description

Check if EXTI line 3 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE6 SR_EXTI3 LL_SYSCFG_IsActiveFlag_EXTI3

LL_SYSCFG_IsActiveFlag_EXTI4

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI4 (void)

Function description

Check if EXTI line 4 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI4 LL_SYSCFG_IsActiveFlag_EXTI4

LL_SYSCFG_IsActiveFlag_EXTI5

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI5 (void)

Function description

Check if EXTI line 5 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI5 LL_SYSCFG_IsActiveFlag_EXTI5

LL_SYSCFG_IsActiveFlag_EXTI6

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI6 (void)

Function description

Check if EXTI line 6 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI6 LL_SYSCFG_IsActiveFlag_EXTI6

LL_SYSCFG_IsActiveFlag_EXTI7

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI7 (void)

Function description

Check if EXTI line 7 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI7 LL_SYSCFG_IsActiveFlag_EXTI7

LL_SYSCFG_IsActiveFlag_EXTI8

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI8 (void)

Function description

Check if EXTI line 8 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI8 LL_SYSCFG_IsActiveFlag_EXTI8

LL_SYSCFG_IsActiveFlag_EXTI9

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI9 (void)

Function description

Check if EXTI line 9 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI9 LL_SYSCFG_IsActiveFlag_EXTI9

LL_SYSCFG_IsActiveFlag_EXTI10

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI10 (void)

Function description

Check if EXTI line 10 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI10 LL_SYSCFG_IsActiveFlag_EXTI10

LL_SYSCFG_IsActiveFlag_EXTI11

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI11 (void)

Function description

Check if EXTI line 11 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI11 LL_SYSCFG_IsActiveFlag_EXTI11

LL_SYSCFG_IsActiveFlag_EXTI12

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI12 (void)

Function description

Check if EXTI line 12 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI12 LL_SYSCFG_IsActiveFlag_EXTI12

LL_SYSCFG_IsActiveFlag_EXTI13

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI13 (void)

Function description

Check if EXTI line 13 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI13 LL_SYSCFG_IsActiveFlag_EXTI13

LL_SYSCFG_IsActiveFlag_EXTI14

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI14 (void)

Function description

Check if EXTI line 14 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI14 LL_SYSCFG_IsActiveFlag_EXTI14

LL_SYSCFG_IsActiveFlag_EXTI15

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_EXTI15 (void)

Function description

Check if EXTI line 15 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE7 SR_EXTI15 LL_SYSCFG_IsActiveFlag_EXTI15

LL_SYSCFG_IsActiveFlag_DMA1_CH1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH1 (void)

Function description

Check if DMA1 channel 1 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE9 SR_DMA1_CH1 LL_SYSCFG_IsActiveFlag_DMA1_CH1

LL_SYSCFG_IsActiveFlag_DMA1_CH2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH2 (void)

Function description

Check if DMA1 channel 2 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE10 SR_DMA1_CH2 LL_SYSCFG_IsActiveFlag_DMA1_CH2

LL_SYSCFG_IsActiveFlag_DMA1_CH3

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH3 (void)

Function description

Check if DMA1 channel 3 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE10 SR_DMA1_CH3 LL_SYSCFG_IsActiveFlag_DMA1_CH3

LL_SYSCFG_IsActiveFlag_DMAMUX

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMAMUX (void)

Function description

Check if DMAMUX interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE11 SR_DMAMUX1 LL_SYSCFG_IsActiveFlag_DMAMUX

LL_SYSCFG_IsActiveFlag_DMA1_CH4

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH4 (void)

Function description

Check if DMA1_CH4 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE11 SR_DMA1_CH4 LL_SYSCFG_IsActiveFlag_DMA1_CH4

LL_SYSCFG_IsActiveFlag_DMA1_CH5

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH5 (void)

Function description

Check if DMA1_CH5 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE11 SR_DMA1_CH5 LL_SYSCFG_IsActiveFlag_DMA1_CH5

LL_SYSCFG_IsActiveFlag_DMA1_CH6

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH6 (void)

Function description

Check if DMA1_CH6 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE11 SR_DMA1_CH6 LL_SYSCFG_IsActiveFlag_DMA1_CH6

LL_SYSCFG_IsActiveFlag_DMA1_CH7

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_DMA1_CH7 (void)

Function description

Check if DMA1_CH7 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE11 SR_DMA1_CH7 LL_SYSCFG_IsActiveFlag_DMA1_CH7

LL_SYSCFG_IsActiveFlag_ADC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_ADC (void)

Function description

Check if ADC interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE12 SR_ADC LL_SYSCFG_IsActiveFlag_ADC

LL_SYSCFG_IsActiveFlag_TIM1_BRK

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_BRK (void)

Function description

Check if Timer 1 break interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_BRK LL_SYSCFG_IsActiveFlag_TIM1_BRK

LL_SYSCFG_IsActiveFlag_TIM1_UPD

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_UPD (void)

Function description

Check if Timer 1 update interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_UPD LL_SYSCFG_IsActiveFlag_TIM1_UPD

LL_SYSCFG_IsActiveFlag_TIM1_TRG

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_TRG (void)

Function description

Check if Timer 1 trigger interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_TRG LL_SYSCFG_IsActiveFlag_TIM1_TRG

LL_SYSCFG_IsActiveFlag_TIM1_CCU

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_CCU (void)

Function description

Check if Timer 1 commutation interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE13 SR_TIM1_CCU LL_SYSCFG_IsActiveFlag_TIM1_CCU

LL_SYSCFG_IsActiveFlag_TIM1_CC

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM1_CC (void)

Function description

Check if Timer 1 capture compare interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE14 SR_TIM1_CC LL_SYSCFG_IsActiveFlag_TIM1_CC

LL_SYSCFG_IsActiveFlag_TIM2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM2 (void)

Function description

Check if Timer 2 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE15 SR_TIM2_GLB LL_SYSCFG_IsActiveFlag_TIM2

LL_SYSCFG_IsActiveFlag_TIM3

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM3 (void)

Function description

Check if Timer 3 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE16 SR_TIM3_GLB LL_SYSCFG_IsActiveFlag_TIM3

LL_SYSCFG_IsActiveFlag_TIM14

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM14 (void)

Function description

Check if Timer 14 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE19 SR_TIM14_GLB LL_SYSCFG_IsActiveFlag_TIM14

LL_SYSCFG_IsActiveFlag_TIM15

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM15 (void)

Function description

Check if Timer 15 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE20 SR_TIM15_GLB LL_SYSCFG_IsActiveFlag_TIM15

LL_SYSCFG_IsActiveFlag_TIM16

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM16 (void)

Function description

Check if Timer 16 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE21 SR_TIM16_GLB LL_SYSCFG_IsActiveFlag_TIM16

LL_SYSCFG_IsActiveFlag_TIM17

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_TIM17 (void)

Function description

Check if Timer 17 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE22 SR_TIM17_GLB LL_SYSCFG_IsActiveFlag_TIM17

LL_SYSCFG_IsActiveFlag_I2C1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_I2C1 (void)

Function description

Check if I2C1 interrupt occurred or not, combined with EXTI line 23.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE23 SR_I2C1_GLB LL_SYSCFG_IsActiveFlag_I2C1

LL_SYSCFG_IsActiveFlag_I2C2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_I2C2 (void)

Function description

Check if I2C2 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE24 SR_I2C2_GLB LL_SYSCFG_IsActiveFlag_I2C2

LL_SYSCFG_IsActiveFlag_SPI1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SPI1 (void)

Function description

Check if SPI1 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE25 SR_SPI1 LL_SYSCFG_IsActiveFlag_SPI1

LL_SYSCFG_IsActiveFlag_SPI2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_SPI2 (void)

Function description

Check if SPI2 interrupt occurred or not.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE26_SR_SPI2 LL_SYSCFG_IsActiveFlag_SPI2

LL_SYSCFG_IsActiveFlag_USART1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART1 (void)

Function description

Check if USART1 interrupt occurred or not, combined with EXTI line 25.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE27 SR_USART1_GLB LL_SYSCFG_IsActiveFlag_USART1

LL_SYSCFG_IsActiveFlag_USART2

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART2 (void)

Function description

Check if USART2 interrupt occurred or not, combined with EXTI line 26.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE28 SR_USART2_GLB LL_SYSCFG_IsActiveFlag_USART2

LL_SYSCFG_IsActiveFlag_USART3

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART3 (void)

Function description

Check if USART3 interrupt occurred or not, combined with EXTI line 26.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART3_GLB LL_SYSCFG_IsActiveFlag_USART3

LL_SYSCFG_IsActiveFlag_USART4

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_USART4 (void)

Function description

Check if USART4 interrupt occurred or not, combined with EXTI line 26.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE29 SR_USART4_GLB LL_SYSCFG_IsActiveFlag_USART4

LL_SYSCFG_IsActiveFlag_FDCAN1_IT0

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_FDCAN1_IT0 (void)

Function description

Check if FDCAN1_IT0 interrupt occurred or not, combined with EXTI line 26.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE30 SR_FDCAN1_IT0 LL_SYSCFG_IsActiveFlag_FDCAN1_IT0

LL_SYSCFG_IsActiveFlag_FDCAN1_IT1

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_IsActiveFlag_FDCAN1_IT1 (void)

Function description

Check if FDCAN1_IT1 interrupt occurred or not, combined with EXTI line 26.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- SYSCFG_ITLINE31 SR_FDCAN1_IT1 LL_SYSCFG_IsActiveFlag_FDCAN1_IT1

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=0xFFF

Reference Manual to LL API cross reference:

- DBG_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:

- DBG_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:

- DBG_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:

- DBG_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:

- DBG_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:

- DBG_CR DBG_STANDBY LL_DBGMCU_DisableDBGStandbyMode

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_RTC_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBG_APB_FZ1 DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBG_APB_FZ1 DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBG_APB_FZ1 DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBG_APB_FZ1 DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_FreezePeriph
- DBG_APB_FZ1 DBG_I2C1_STOP LL_DBGMCU_APB1_GRP1_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_RTC_STOP
  - LL_DBGMCU_APB1_GRP1_WWDG_STOP
  - LL_DBGMCU_APB1_GRP1_IWDG_STOP
  - LL_DBGMCU_APB1_GRP1_I2C1_STOP

  (*) value not defined in all devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBG_APB_FZ1 DBG_TIM3_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBG_APB_FZ1 DBG_TIM2_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBG_APB_FZ1 DBG_RTC_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBG_APB_FZ1 DBG_WWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBG_APB_FZ1 DBG_IWDG_STOP LL_DBGMCU_APB1_GRP1_UnFreezePeriph
- DBG_APB_FZ1 DBG_I2C1_SMBUS_TIMEOUT LL_DBGMCU_APB1_GRP1_UnFreezePeriph
-

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_TIM1_STOP
  - LL_DBGMCU_APB1_GRP2_TIM14_STOP
  - LL_DBGMCU_APB1_GRP2_TIM16_STOP
  - LL_DBGMCU_APB1_GRP2_TIM17_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBG_APB_FZ2 DBG_TIM1_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph
- DBG_APB_FZ2 DBG_TIM14_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph
- DBG_APB_FZ2 DBG_TIM16_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph
- DBG_APB_FZ2 DBG_TIM17_STOP LL_DBGMCU_APB1_GRP2_FreezePeriph

LL_DBGMCU_APB1_GRP2_UnFreezePeriph

Function name

__STATIC_INLINE void LL_DBGMCU_APB1_GRP2_UnFreezePeriph (uint32_t Periphs)

Function description

Unfreeze APB2 peripherals.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_DBGMCU_APB1_GRP2_TIM1_STOP
  - LL_DBGMCU_APB1_GRP2_TIM14_STOP
  - LL_DBGMCU_APB1_GRP2_TIM16_STOP
  - LL_DBGMCU_APB1_GRP2_TIM17_STOP

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBG_APB_FZ2 DBG_TIM1_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph
- DBG_APB_FZ2 DBG_TIM14_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph
- DBG_APB_FZ2 DBG_TIM16_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph
- DBG_APB_FZ2 DBG_TIM17_STOP LL_DBGMCU_APB1_GRP2_UnFreezePeriph

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

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR 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

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_LATENCY LL_FLASH_GetLatency

LL_FLASH_EnablePrefetch

Function name

__STATIC_INLINE void LL_FLASH_EnablePrefetch (void)

Function description

Enable Prefetch.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_PRFTEN LL_FLASH_EnablePrefetch

LL_FLASH_DisablePrefetch

Function name

__STATIC_INLINE void LL_FLASH_DisablePrefetch (void)

Function description

Disable Prefetch.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_PRFTEN LL_FLASH_DisablePrefetch

LL_FLASH_IsPrefetchEnabled

Function name

__STATIC_INLINE uint32_t LL_FLASH_IsPrefetchEnabled (void)

Function description

Check if Prefetch buffer is enabled.

Return values

- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_PRFTEN LL_FLASH_IsPrefetchEnabled

LL_FLASH_EnableInstCache

Function name

__STATIC_INLINE void LL_FLASH_EnableInstCache (void)

Function description

Enable Instruction cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_ICEN LL_FLASH_EnableInstCache

LL_FLASH_DisableInstCache

Function name

__STATIC_INLINE void LL_FLASH_DisableInstCache (void)

Function description

Disable Instruction cache.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_ICEN LL_FLASH_DisableInstCache

LL_FLASH_EnableInstCacheReset

Function name

__STATIC_INLINE void LL_FLASH_EnableInstCacheReset (void)

Function description

Enable Instruction cache reset.

Return values

- **None:**

Notes

- bit can be written only when the instruction cache is disabled

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_ICRST LL_FLASH_EnableInstCacheReset

LL_FLASH_DisableInstCacheReset

Function name

__STATIC_INLINE void LL_FLASH_DisableInstCacheReset (void)

Function description

Disable Instruction cache reset.

Return values

- **None:**

Reference Manual to LL API cross reference:

- FLASH_ACR FLASH_ACR_ICRST LL_FLASH_DisableInstCacheReset

59.2 SYSTEM Firmware driver defines

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

59.2.1 SYSTEM

SYSTEM

DBGMCU APB1 GRP1 STOP IP

LL_DBGMCU_APB1_GRP1_TIM2_STOP

TIM2 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_TIM3_STOP

TIM3 counter stopped when core is halted

LL_DBGMCU_APB1_GRP1_RTC_STOP

RTC Calendar frozen when core is halted

LL_DBGMCU_APB1_GRP1_WWDG_STOP

Debug Window Watchdog stopped when Core is halted

LL_DBGMCU_APB1_GRP1_IWDG_STOP

Debug Independent Watchdog stopped when Core is halted

LL_DBGMCU_APB1_GRP1_I2C1_STOP

I2C1 SMBUS timeout mode stopped when Core is halted

DBGMCU APB1 GRP2 STOP IP

LL_DBGMCU_APB1_GRP2_TIM1_STOP

TIM1 counter stopped when core is halted

LL_DBGMCU_APB1_GRP2_TIM14_STOP

TIM14 counter stopped when core is halted

LL_DBGMCU_APB1_GRP2_TIM16_STOP

TIM16 counter stopped when core is halted

LL_DBGMCU_APB1_GRP2_TIM17_STOP

TIM17 counter stopped when core is halted

LL_DBGMCU_APB2_GRP1_TIM1_STOP

LL_DBGMCU_APB2_GRP1_TIM14_STOP

LL_DBGMCU_APB2_GRP1_TIM16_STOP

LL_DBGMCU_APB2_GRP1_TIM17_STOP

SYSCFG I/O analog switch voltage booster enable

LL_SYSCFG_CFGR1_BOOSTEN

I/O analog switch voltage booster enable

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_PB6

I2C PB6 Fast mode plus

LL_SYSCFG_I2C_FASTMODEPLUS_PB7

I2C PB7 Fast mode plus

LL_SYSCFG_I2C_FASTMODEPLUS_PB8

I2C PB8 Fast mode plus

LL_SYSCFG_I2C_FASTMODEPLUS_PB9

I2C PB9 Fast mode plus

LL_SYSCFG_I2C_FASTMODEPLUS_I2C1

Enable I2C1 Fast mode Plus

LL_SYSCFG_I2C_FASTMODEPLUS_I2C2

Enable I2C2 Fast mode Plus

LL_SYSCFG_I2C_FASTMODEPLUS_PA9

Enable Fast Mode Plus on PA9

LL_SYSCFG_I2C_FASTMODEPLUS_PA10

Enable Fast Mode Plus on PA10

LL_SYSCFG_I2C_FASTMODEPLUS_PC14

Enable Fast Mode Plus on PC14

SYSCFG IR Modulation

LL_SYSCFG_IR_MOD_TIM16

00: Timer16 is selected as IRDA Modulation envelope source

LL_SYSCFG_IR_MOD_USART1

01: USART1 is selected as IRDA Modulation envelope source

LL_SYSCFG_IR_MOD_USART2

10: USART2 is selected as IRDA Modulation envelope source

SYSCFG IR Polarity

LL_SYSCFG_IR_POL_NOT_INVERTED

0: Output of IRDA (IROut) not inverted

LL_SYSCFG_IR_POL_INVERTED

1: Output of IRDA (IROut) inverted

FLASH LATENCY

LL_FLASH_LATENCY_0

FLASH Zero Latency cycle

LL_FLASH_LATENCY_1

FLASH One Latency cycle

SYSCFG PIN RMP

LL_SYSCFG_PIN_RMP_PA11

PA11 pad behaves as PA9 pin

LL_SYSCFG_PIN_RMP_PA12

PA12 pad behaves as PA10 pin

SYSCFG REMAP

LL_SYSCFG_REMAP_FLASH

Main Flash memory mapped at 0x00000000

LL_SYSCFG_REMAP_SYSTEMFLASH

System Flash memory mapped at 0x00000000

LL_SYSCFG_REMAP_SRAM

Embedded SRAM mapped at 0x00000000

SYSCFG TIMER BREAK

LL_SYSCFG_TIMBREAK_LOCKUP

Enables and locks the LOCKUP (Hardfault) output of CortexM0 with Break Input of TIM1/16/17

DBGMCU

LL_DBGMCU_APB2_GRP1_FreezePeriph

LL_DBGMCU_APB2_GRP1_UnFreezePeriph

PINMUX Config

LL_PINMUX_TSSOP20_PIN19_PA14

STM32C091/92 TSSOP20 package, Pin19 assigned to GPIO PA14

LL_PINMUX_TSSOP20_PIN19_PA15

STM32C091/92 TSSOP20 package, Pin19 assigned to GPIO PA15

LL_PINMUX_TSSOP20_PIN15_PA8

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PA8

LL_PINMUX_TSSOP20_PIN15_PB0

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PB0

LL_PINMUX_TSSOP20_PIN15_PB1

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PB1

LL_PINMUX_TSSOP20_PIN15_PB2

STM32C091/92 TSSOP20 package, Pin15 assigned to GPIO PB2

LL_PINMUX_WLCSP24_PING1_PA8

LL_PINMUX_WLCSP24_PING1_PB2

LL_PINMUX_TSSOP20_PIN20_PB6

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB6

LL_PINMUX_TSSOP20_PIN20_PB3

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB3

LL_PINMUX_TSSOP20_PIN20_PB4

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB4

LL_PINMUX_TSSOP20_PIN20_PB5

STM32C091/92 TSSOP20 package, Pin20 assigned to GPIO PB5

LL_PINMUX_WLCSP24_PINA5_PB6

LL_PINMUX_WLCSP24_PINA5_PB4

LL_PINMUX_WLCSP24_PINA3_PB5

STM32C091/92 WLCSP24 package, PinA3 assigned to GPIO PB5

LL_PINMUX_WLCSP24_PINA3_PB3

STM32C091/92 WLCSP24 package, PinA3 assigned to GPIO PB3

LL_PINMUX_WLCSP24_PINB4_PB7

STM32C091/92 WLCSP24 package, PinB4 assigned to GPIO PB7

LL_PINMUX_WLCSP24_PINB4_PB8

STM32C091/92 WLCSP24 package, PinB4 assigned to GPIO PB8

LL_PINMUX_TSSOP20_PIN1_PB7

LL_PINMUX_TSSOP20_PIN1_PB8

PINMUX Config Source

LL_PINMUX_TSSOP20_PIN19

STM32C091/92 TSSOP20 package, GPIO Pin19 multiplexer

LL_PINMUX_TSSOP20_PIN15

STM32C091/92 TSSOP20 package, GPIO Pin15 multiplexer

LL_PINMUX_WLCSP24_PING1

LL_PINMUX_TSSOP20_PIN20

STM32C091/92 TSSOP20 package, GPIO Pin20 multiplexer

LL_PINMUX_WLCSP24_PINA5

LL_PINMUX_WLCSP24_PINA3

STM32C091/92 WLCSP24 package, GPIO PinA3 multiplexer

LL_PINMUX_WLCSP24_PINB4

STM32C091/92 WLCSP24 package, GPIO PinB4 multiplexer

LL_PINMUX_TSSOP20_PIN1

60 LL TIM Generic Driver

60.1 TIM Firmware driver registers structures

60.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32c0xx_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()**.

60.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32c0xx_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()**.

60.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32c0xx_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()**.

60.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32c0xx_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()**.

60.1.5 LL_TIM_HALLSENSOR_InitTypeDef

LL_TIM_HALLSENSOR_InitTypeDef is defined in the stm32c0xx_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()**.

60.1.6 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32c0xx_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.

60.2 TIM Firmware driver API description

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

60.2.1 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.

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.

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

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.

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_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 (const 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_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 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_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 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_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 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

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

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_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 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_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 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_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 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_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 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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_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.

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.

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.

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.

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

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

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

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_ITR7 (*)
  - LL_TIM_TS_TI1F_ED
  - LL_TIM_TS_TI1FP1
  - LL_TIM_TS_TI2FP2
  - LL_TIM_TS_ETRF

  (*) Value not defined in all devices.

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: TIM1

Return values

- **None:**

Notes

- Macro IS_TIM_ETRSEL_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_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

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
- AF2 BK2INE 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

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
- AF2 BK2INE 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
- **Polarity:** This parameter can be one of the following values:
  - LL_TIM_BKIN_POLARITY_LOW
  - LL_TIM_BKIN_POLARITY_HIGH

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
- AF2 BK2INP LL_TIM_SetBreakInputSourcePolarity
-

LL_TIM_ConfigDMABurst

Function name

__STATIC_INLINE void LL_TIM_ConfigDMABurst (TIM_TypeDef * TIMx, uint32_t DMABurstBaseAddress, uint32_t DMABurstLength)

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_CCMR3
  - LL_TIM_DMABURST_BASEADDR_CCR5
  - LL_TIM_DMABURST_BASEADDR_CCR6
  - LL_TIM_DMABURST_BASEADDR_AF1
  - LL_TIM_DMABURST_BASEADDR_AF2
  - LL_TIM_DMABURST_BASEADDR_TISEL
- **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

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_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
- TIM1_TISEL TI2SEL LL_TIM_SetRemap
- TIM1_TISEL TI3SEL LL_TIM_SetRemap
- TIM1_TISEL TI4SEL LL_TIM_SetRemap
- TIM2_TISEL TI1SEL LL_TIM_SetRemap
- TIM2_TISEL TI2SEL LL_TIM_SetRemap
- TIM2_TISEL TI3SEL LL_TIM_SetRemap
- TIM2_TISEL TI4SEL LL_TIM_SetRemap
- TIM3_TISEL TI1SEL LL_TIM_SetRemap
- TIM3_TISEL TI2SEL LL_TIM_SetRemap
- TIM3_TISEL TI3SEL LL_TIM_SetRemap
- TIM3_TISEL TI4SEL LL_TIM_SetRemap
- TIM14_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_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

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

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_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_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.

60.3 TIM Firmware driver defines

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

60.3.1 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

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_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_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_CCMR3

TIMx_CCMR3 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_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_TISEL

TIMx_TISEL 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

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

External Trigger Source

LL_TIM_ETRSOURCE_GPIO

ETR input is connected to GPIO

LL_TIM_ETRSOURCE_ADC1_AWD1

ETR input is connected to ADC1 analog watchdog 1

LL_TIM_ETRSOURCE_ADC1_AWD2

ETR input is connected to ADC1 analog watchdog 2

LL_TIM_ETRSOURCE_ADC1_AWD3

ETR input is connected to ADC1 analog watchdog 3

LL_TIM_ETRSOURCE_LSE

ETR input is connected to LSE

LL_TIM_ETRSOURCE_MCO

ETR input is connected to MCO

LL_TIM_ETRSOURCE_MCO2

ETR input is connected to MCO2

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

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

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

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

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

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

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

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

TIM14 Timer Input Ch1 Remap

LL_TIM_TIM14_TI1_RMP_GPIO

TIM14 input 1 is connected to GPIO

LL_TIM_TIM14_TI1_RMP_RTC_CLK

TIM14 input 1 is connected to RTC clock

LL_TIM_TIM14_TI1_RMP_HSE_32

TIM14 input 1 is connected to HSE/32 clock

LL_TIM_TIM14_TI1_RMP_MCO

TIM14 input 1 is connected to MCO

LL_TIM_TIM14_TI1_RMP_MCO2

TIM14 input 1 is connected to MCO2

TIM15 Timer Input Ch1 Remap

LL_TIM_TIM15_TI1_RMP_GPIO

TIM15 input 1 is connected to GPIO

LL_TIM_TIM15_TI1_RMP_TIM2_IC1

TIM15 input 1 is connected to TIM2 input 1

LL_TIM_TIM15_TI1_RMP_TIM3_IC1

TIM15 input 1 is connected to TIM3 input 1

TIM15 Timer Input Ch2 Remap

LL_TIM_TIM15_TI2_RMP_GPIO

TIM15 input 2 is connected to GPIO

LL_TIM_TIM15_TI2_RMP_TIM2_IC2

TIM15 input 2 is connected to TIM2 input 2

LL_TIM_TIM15_TI2_RMP_TIM3_IC2

TIM15 input 2 is connected to TIM3 input 2

TIM16 Timer Input Ch1 Remap

LL_TIM_TIM16_TI1_RMP_GPIO

TIM16 input 1 is connected to GPIO

LL_TIM_TIM16_TI1_RMP_LSI

TIM16 input 1 is connected to LSI

LL_TIM_TIM16_TI1_RMP_LSE

TIM16 input 1 is connected to LSE

LL_TIM_TIM16_TI1_RMP_MCO2

TIM16 input 1 is connected to MCO2

TIM17 Timer Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17 input 1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_HSE_32

TIM17 input 1 is connected to HSE/32 clock

LL_TIM_TIM17_TI1_RMP_MCO

TIM17 input 1 is connected to MCO

LL_TIM_TIM17_TI1_RMP_MCO2

TIM17 input 1 is connected to MCO2

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

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_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_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_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_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 ());

61 LL USART Generic Driver

61.1 USART Firmware driver registers structures

61.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32c0xx_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()**.

61.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32c0xx_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.

61.2 USART Firmware driver API description

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

61.2.1 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_SetWKUPType

Function name

__STATIC_INLINE void LL_USART_SetWKUPType (USART_TypeDef * USARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance
- **Type:** This parameter can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

Return values

- **None:**

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:

- CR3 WUS LL_USART_SetWKUPType

LL_USART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_USART_GetWKUPType (const USART_TypeDef * USARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **USARTx:** USART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_USART_WAKEUP_ON_ADDRESS
  - LL_USART_WAKEUP_ON_STARTBIT
  - LL_USART_WAKEUP_ON_RXNE

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:

- CR3 WUS LL_USART_GetWKUPType

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_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsActiveFlag_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from stop mode Flag is set 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:

- ISR WUF LL_USART_IsActiveFlag_WKUP

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_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_USART_ClearFlag_WKUP (USART_TypeDef * USARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

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:

- ICR WUCF LL_USART_ClearFlag_WKUP

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_WKUP

Function name

__STATIC_INLINE void LL_USART_EnableIT_WKUP (USART_TypeDef * USARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

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:

- CR3 WUFIE LL_USART_EnableIT_WKUP

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_WKUP

Function name

__STATIC_INLINE void LL_USART_DisableIT_WKUP (USART_TypeDef * USARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **USARTx:** USART Instance

Return values

- **None:**

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:

- CR3 WUFIE LL_USART_DisableIT_WKUP

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_WKUP

Function name

__STATIC_INLINE uint32_t LL_USART_IsEnabledIT_WKUP (const USART_TypeDef * USARTx)

Function description

Check if the USART Wake Up from Stop Mode Interrupt is enabled or disabled.

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:

- CR3 WUFIE LL_USART_IsEnabledIT_WKUP

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_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:**

61.3 USART Firmware driver defines

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

61.3.1 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

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

LL_USART_ICR_WUCF

Wakeup from Stop mode 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_WUF

Wakeup from Stop 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_WUFIE

Wakeup from Stop mode 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

Wakeup Activation

LL_USART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_USART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_USART_WAKEUP_ON_RXNE

Wake up active on RXNE

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

62 LL UTILS Generic Driver

62.1 UTILS Firmware driver API description

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

62.1.1 System Configuration functions

System, AHB and APB buses clocks configuration

- The maximum frequency of the SYSCLK, HCLK, PCLK1 is 48000000 Hz.

This section contains the following APIs:

- *LL_SetSystemCoreClock()*
- *LL_SetFlashLatency()*

62.1.2 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_SO8 (*)
  - LL_UTILS_PACKAGETYPE_WLCSP12 (*)
  - LL_UTILS_PACKAGETYPE_UFQFPN20 (*)
  - LL_UTILS_PACKAGETYPE_TSSOP20 (*)
  - LL_UTILS_PACKAGETYPE_UFQFPN28 (*)
  - LL_UTILS_PACKAGETYPE_QFN32 (*)
  - LL_UTILS_PACKAGETYPE_QFN48 (*)
  - LL_UTILS_PACKAGETYPE_WLCSP19 (*)
  - LL_UTILS_PACKAGETYPE_WLCSP15 (*)
  - LL_UTILS_PACKAGETYPE_WLCSP24 (*)
  - LL_UTILS_PACKAGETYPE_UFBGA64 (*)
  - LL_UTILS_PACKAGETYPE_LQFP64 (*)

Notes

- (*) Availability depends on devices.

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.

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_mDelay

Function name

void LL_mDelay (uint32_t Delay)

Function description

This function provides accurate delay (in milliseconds) based on SysTick counter flag.

Parameters

- **Delay:** specifies the 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_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

62.2 UTILS Firmware driver defines

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

62.2.1 UTILS

UTILS

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_TSSOP20

TSSOP20 package type

LL_UTILS_PACKAGETYPE_WLCSP24

WLCSP24 package type

LL_UTILS_PACKAGETYPE_UFQFPN28

UFQFPN28 package type

LL_UTILS_PACKAGETYPE_QFN32

UFQFPN32 / LQFP32 package type

LL_UTILS_PACKAGETYPE_QFN48

UFQFPN48 / LQFP48 package type

LL_UTILS_PACKAGETYPE_QFN64

UFQFPN64 / LQFP64 package type

SYSTEM

UTILS_SetFlashLatency

63 LL WWDG Generic Driver

63.1 WWDG Firmware driver API description

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

63.1.1 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

63.2 WWDG Firmware driver defines

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

63.2.1 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

64 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 STM32C0 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: stm32c0xxxx_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 (stm32c0xxxx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32c0xxxx_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 (stm32c0xxxx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32c0xxxx_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_stm32c0xxxx.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 stm32c0xxxx_hal_msp.c. A template is provided in the HAL driver folders (stm32c0xxxx_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 stm32c0xxxx_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 stm32c0xxxx_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 stm32c0xxxx_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 stm32c0xxxx_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 25. **Document revision history**

Date	Revision	Changes
09-Sep-2022	1	Initial release.
21-Aug-2024	2	Updated list of supported devices in Section 3.4: Devices supported by HAL/LL drivers . Removed description of status bit and flag from Section 3.5.2: HAL general naming rules. Removed section *Updating existing common APIs* from Section 3.7.2: HAL extension model cases. Updated code examples in Section 3.12.4.1: Timeout management. Added HAL HCD Generic Driver. Added HAL PCD Generic Driver and HAL PCD Extension Driver.
29-Jan-2025	3	Updated list of supported devices in Section 3.4: Devices supported by HAL/LL drivers . Added HAL FDCAN driver.

Contents

1 General information 2
2 Acronyms and definitions. 3
3 Overview of HAL drivers 5
3.1 HAL and user-application files 6
3.1.1 HAL driver files 6
3.1.2 User-application files 6
3.2 HAL data structures 8
3.2.1 Peripheral handle structures 8
3.2.2 Initialization and configuration structure 10
3.2.3 Specific process structures 10
3.3 API classification 11
3.4 Devices supported by HAL/LL drivers 12
3.5 HAL driver rules. 14
3.5.1 HAL API naming rules. 14
3.5.2 HAL general naming rules. 15
3.5.3 HAL interrupt handler and callback functions 16
3.6 HAL generic APIs 16
3.7 HAL extension APIs 17
3.7.1 HAL extension model overview 17
3.7.2 HAL extension model cases 18
3.8 File inclusion model 19
3.9 HAL common resources 20
3.10 HAL configuration 20
3.11 HAL system peripheral handling 21
3.11.1 Clocks 21
3.11.2 GPIOs 22
3.11.3 Cortex® NVIC and SysTick timer. 23
3.11.4 PWR 23
3.11.5 EXTI 24
3.11.6 DMA 25
3.12 How to use HAL drivers 25
3.12.1 HAL usage models 25
3.12.2 HAL initialization 26
3.12.3 HAL I/O operation process 28
3.12.4 Timeout and error management 31

4 Overview of low-layer drivers 35
4.1 Low-layer files 35
4.2 Overview of low-layer APIs and naming rules 37
4.2.1 Peripheral initialization functions 37
4.2.2 Peripheral register-level configuration functions 39
5 Cohabiting of HAL and LL 41
5.1 Low-layer driver used in standalone mode 41
5.2 Mixed use of low-layer APIs and HAL drivers 41
6 HAL System Driver 42
6.1 HAL Firmware driver API description 42
6.1.1 How to use this driver 42
6.1.2 HAL Initialization and Configuration functions 42
6.1.3 HAL Control functions 42
6.1.4 HAL Debug functions 43
6.1.5 HAL SYSCFG configuration functions 43
6.1.6 Detailed description of functions 43
6.2 HAL Firmware driver defines 50
6.2.1 HAL 50
7 HAL ADC Generic Driver 60
7.1 ADC Firmware driver registers structures 60
7.1.1 ADC_OversamplingTypeDef 60
7.1.2 ADC_InitTypeDef 60
7.1.3 ADC_ChannelConfTypeDef 63
7.1.4 ADC_AnalogWDGConfTypeDef 64
7.1.5 __ADC_HandleTypeDef 64
7.2 ADC Firmware driver API description 66
7.2.1 ADC peripheral features 66
7.2.2 How to use this driver 66
7.2.3 Peripheral Control functions 68
7.2.4 Peripheral state and errors functions 69
7.2.5 Detailed description of functions 69
7.3 ADC Firmware driver defines 79
7.3.1 ADC 79
8 HAL ADC Extension Driver 102
8.1 ADCEx Firmware driver API description 102
8.1.1 IO operation functions 102
8.1.2 Detailed description of functions 102

8.2 ADCEx Firmware driver defines . . . 104
8.2.1 ADCEx . . . 104
9 HAL CORTEX Generic Driver . . . 105
9.1 CORTEX Firmware driver registers structures . . . 105
9.1.1 MPU_Region_InitTypeDef . . . 105
9.2 CORTEX Firmware driver API description . . . 105
9.2.1 How to use this driver . . . 106
9.2.2 Initialization and Configuration functions . . . 106
9.2.3 Peripheral Control functions . . . 106
9.2.4 Detailed description of functions . . . 107
9.3 CORTEX Firmware driver defines . . . 111
9.3.1 CORTEX . . . 111
10 HAL CRC Generic Driver . . . 115
10.1 CRC Firmware driver registers structures . . . 115
10.1.1 CRC_InitTypeDef . . . 115
10.1.2 CRC_HandleTypeDef . . . 116
10.2 CRC Firmware driver API description . . . 116
10.2.1 How to use this driver . . . 116
10.2.2 Initialization and de-initialization functions . . . 116
10.2.3 Peripheral Control functions . . . 117
10.2.4 Peripheral State functions . . . 117
10.2.5 Detailed description of functions . . . 117
10.3 CRC Firmware driver defines . . . 119
10.3.1 CRC . . . 119
11 HAL CRC Extension Driver . . . 122
11.1 CRCEx Firmware driver API description . . . 122
11.1.1 How to use this driver . . . 122
11.1.2 Extended configuration functions . . . 122
11.1.3 Detailed description of functions . . . 122
11.2 CRCEx Firmware driver defines . . . 123
11.2.1 CRCEx . . . 123
12 HAL DMA Generic Driver . . . 125
12.1 DMA Firmware driver registers structures . . . 125
12.1.1 DMA_InitTypeDef . . . 125
12.1.2 __DMA_HandleTypeDef . . . 125
12.2 DMA Firmware driver API description . . . 126
12.2.1 How to use this driver . . . 126

12.2.2 Initialization and de-initialization functions . . . 127
12.2.3 IO operation functions . . . 127
12.2.4 Peripheral State and Errors functions . . . 128
12.2.5 Detailed description of functions . . . 128
12.3 DMA Firmware driver defines . . . 131
12.3.1 DMA . . . 131
13 HAL DMA Extension Driver . . . 141
13.1 DMAEx Firmware driver registers structures . . . 141
13.1.1 HAL_DMA_MuxSyncConfigTypeDef . . . 141
13.1.2 HAL_DMA_MuxRequestGeneratorConfigTypeDef . . . 141
13.2 DMAEx Firmware driver API description . . . 141
13.2.1 How to use this driver . . . 141
13.2.2 Extended features functions . . . 142
13.2.3 Detailed description of functions . . . 142
13.3 DMAEx Firmware driver defines . . . 143
13.3.1 DMAEx . . . 143
14 HAL EXTI Generic Driver . . . 147
14.1 EXTI Firmware driver registers structures . . . 147
14.1.1 EXTI_HandleTypeDef . . . 147
14.1.2 EXTI_ConfigTypeDef . . . 147
14.2 EXTI Firmware driver API description . . . 147
14.2.1 EXTI Peripheral features . . . 147
14.2.2 How to use this driver . . . 148
14.2.3 Configuration functions . . . 148
14.2.4 Detailed description of functions . . . 148
14.3 EXTI Firmware driver defines . . . 151
14.3.1 EXTI . . . 151
15 HAL FDCAN Generic Driver . . . 154
15.1 FDCAN Firmware driver registers structures . . . 154
15.1.1 FDCAN_InitTypeDef . . . 154
15.1.2 FDCAN_FilterTypeDef . . . 155
15.1.3 FDCAN_TxHeaderTypeDef . . . 155
15.1.4 FDCAN_RxHeaderTypeDef . . . 156
15.1.5 FDCAN_TxEventFifoTypeDef . . . 157
15.1.6 FDCAN_HpMsgStatusTypeDef . . . 158
15.1.7 FDCAN_ProtocolStatusTypeDef . . . 158
15.1.8 FDCAN_ErrorCountersTypeDef . . . 159

15.1.9 FDCAN_MsgRamAddressTypeDef . . . 160
15.1.10 __FDCAN_HandleTypeDef . . . 160
15.2 FDCAN Firmware driver API description . . . 162
15.2.1 How to use this driver . . . 162
15.2.2 Initialization and de-initialization functions . . . 163
15.2.3 Configuration functions . . . 163
15.2.4 Control functions . . . 164
15.2.5 Interrupts management . . . 165
15.2.6 Callback functions . . . 165
15.2.7 Peripheral State functions . . . 166
15.2.8 Detailed description of functions . . . 166
15.3 FDCAN Firmware driver defines . . . 188
15.3.1 FDCAN . . . 188
16 HAL FLASH Generic Driver . . . 201
16.1 FLASH Firmware driver registers structures . . . 201
16.1.1 FLASH_EraseInitTypeDef . . . 201
16.1.2 FLASH_OBProgramInitTypeDef . . . 201
16.1.3 FLASH_ProcessTypeDef . . . 202
16.2 FLASH Firmware driver API description . . . 203
16.2.1 FLASH peripheral features . . . 203
16.2.2 How to use this driver . . . 203
16.2.3 Programming operation functions . . . 203
16.2.4 Peripheral Control functions . . . 204
16.2.5 Peripheral Errors functions . . . 204
16.2.6 Detailed description of functions . . . 204
16.3 FLASH Firmware driver defines . . . 207
16.3.1 FLASH . . . 207
17 HAL FLASH Extension Driver . . . 217
17.1 FLASHEx Firmware driver API description . . . 217
17.1.1 Flash Extended features . . . 217
17.1.2 How to use this driver . . . 217
17.1.3 Extended programming operation functions . . . 217
17.1.4 Detailed description of functions . . . 218
17.2 FLASHEx Firmware driver defines . . . 221
17.2.1 FLASHEx . . . 221
18 HAL GPIO Generic Driver . . . 222
18.1 GPIO Firmware driver registers structures . . . 222

18.1.1 GPIO_InitTypeDef . . . 222
18.2 GPIO Firmware driver API description . . . 222
18.2.1 GPIO Peripheral features . . . 222
18.2.2 How to use this driver . . . 223
18.2.3 Initialization and de-initialization functions . . . 223
18.2.4 Detailed description of functions . . . 223
18.3 GPIO Firmware driver defines . . . 226
18.3.1 GPIO . . . 226
19 HAL GPIO Extension Driver . . . 231
19.1 GPIOEx Firmware driver defines . . . 231
19.1.1 GPIOEx . . . 231
20 HAL I2C Generic Driver . . . 236
20.1 I2C Firmware driver registers structures . . . 236
20.1.1 I2C_InitTypeDef . . . 236
20.1.2 __I2C_HandleTypeDef . . . 236
20.2 I2C Firmware driver API description . . . 238
20.2.1 How to use this driver . . . 238
20.2.2 Initialization and de-initialization functions . . . 243
20.2.3 IO operation functions . . . 244
20.2.4 Peripheral State, Mode and Error functions . . . 245
20.2.5 Detailed description of functions . . . 246
20.3 I2C Firmware driver defines . . . 264
20.3.1 I2C . . . 264
21 HAL I2C Extension Driver . . . 271
21.1 I2CEx Firmware driver API description . . . 271
21.1.1 I2C peripheral Extended features . . . 271
21.1.2 How to use this driver . . . 271
21.1.3 Filter Mode Functions . . . 271
21.1.4 WakeUp Mode Functions . . . 271
21.1.5 Fast Mode Plus Functions . . . 271
21.1.6 Detailed description of functions . . . 272
21.2 I2CEx Firmware driver defines . . . 273
21.2.1 I2CEx . . . 274
22 HAL I2S Generic Driver . . . 275
22.1 I2S Firmware driver registers structures . . . 275
22.1.1 I2S_InitTypeDef . . . 275
22.1.2 __I2S_HandleTypeDef . . . 275

22.2 I2S Firmware driver API description . . . 276
22.2.1 How to use this driver . . . 276
22.2.2 Initialization and de-initialization functions . . . 279
22.2.3 IO operation functions . . . 279
22.2.4 Peripheral State and Errors functions . . . 280
22.2.5 Detailed description of functions . . . 280
22.3 I2S Firmware driver defines . . . 287
22.3.1 I2S . . . 287
23 HAL IRDA Generic Driver . . . 293
23.1 IRDA Firmware driver registers structures . . . 293
23.1.1 IRDA_InitTypeDef . . . 293
23.1.2 __IRDA_HandleTypeDef . . . 293
23.2 IRDA Firmware driver API description . . . 295
23.2.1 How to use this driver . . . 295
23.2.2 Callback registration . . . 296
23.2.3 Initialization and Configuration functions . . . 297
23.2.4 IO operation functions . . . 298
23.2.5 Peripheral State and Error functions . . . 299
23.2.6 Detailed description of functions . . . 299
23.3 IRDA Firmware driver defines . . . 311
23.3.1 IRDA . . . 311
24 HAL IRDA Extension Driver . . . 321
24.1 IRDAEx Firmware driver defines . . . 321
24.1.1 IRDAEx . . . 321
25 HAL IWDG Generic Driver . . . 322
25.1 IWDG Firmware driver registers structures . . . 322
25.1.1 IWDG_InitTypeDef . . . 322
25.1.2 IWDG_HandleTypeDef . . . 322
25.2 IWDG Firmware driver API description . . . 322
25.2.1 IWDG Generic features . . . 322
25.2.2 How to use this driver . . . 323
25.2.3 Initialization and Start functions . . . 323
25.2.4 IO operation functions . . . 323
25.2.5 Detailed description of functions . . . 323
25.3 IWDG Firmware driver defines . . . 324
25.3.1 IWDG . . . 324
26 HAL PWR Generic Driver . . . 326

26.1 PWR Firmware driver API description . . . 326
26.1.1 Initialization and de-initialization functions . . . 326
26.1.2 Peripheral Control functions . . . 326
26.1.3 Detailed description of functions . . . 327
26.2 PWR Firmware driver defines . . . 331
26.2.1 PWR . . . 331
27 HAL PWR Extension Driver . . . 334
27.1 PWREx Firmware driver API description . . . 334
27.1.1 Extended Peripheral Initialization and de-initialization functions . . . 334
27.1.2 Extended PWR Backup register functions . . . 334
27.1.3 Detailed description of functions . . . 334
27.2 PWREx Firmware driver defines . . . 339
27.2.1 PWREx . . . 339
28 HAL RCC Generic Driver . . . 342
28.1 RCC Firmware driver registers structures . . . 342
28.1.1 RCC_OscInitTypeDef . . . 342
28.1.2 RCC_ClkInitTypeDef . . . 342
28.2 RCC Firmware driver API description . . . 343
28.2.1 RCC specific features . . . 343
28.2.2 Initialization and de-initialization functions . . . 343
28.2.3 Peripheral Control functions . . . 344
28.2.4 Detailed description of functions . . . 344
28.3 RCC Firmware driver defines . . . 350
28.3.1 RCC . . . 350
29 HAL RCC Extension Driver . . . 379
29.1 RCCEx Firmware driver registers structures . . . 379
29.1.1 RCC_PeriphCLKInitTypeDef . . . 379
29.2 RCCEx Firmware driver API description . . . 379
29.2.1 Extended Peripheral Control functions . . . 379
29.2.2 Extended clock management functions . . . 379
29.2.3 Detailed description of functions . . . 380
29.3 RCCEx Firmware driver defines . . . 381
29.3.1 RCCEx . . . 381
30 HAL RTC Generic Driver . . . 387
30.1 RTC Firmware driver registers structures . . . 387
30.1.1 RTC_InitTypeDef . . . 387
30.1.2 RTC_TimeTypeDef . . . 387

30.1.3 RTC_DateTypeDef . . . 388
30.1.4 RTC_AlarmTypeDef . . . 388
30.1.5 __RTC_HandleTypeDef . . . 389
30.2 RTC Firmware driver API description . . . 389
30.2.1 How to use RTC Driver . . . 390
30.2.2 RTC and low power modes . . . 390
30.2.3 Initialization and de-initialization functions . . . 390
30.2.4 RTC Time and Date functions . . . 391
30.2.5 RTC Alarm functions . . . 391
30.2.6 Peripheral Control functions . . . 391
30.2.7 Peripheral State functions . . . 392
30.2.8 Detailed description of functions . . . 392
30.3 RTC Firmware driver defines . . . 399
30.3.1 RTC . . . 399
31 HAL RTC Extension Driver . . . 409
31.1 RTCEx Firmware driver API description . . . 409
31.1.1 How to use this driver . . . 409
31.1.2 RTC TimeStamp and Tamper functions . . . 409
31.1.3 Extended Peripheral Control functions . . . 409
31.1.4 Detailed description of functions . . . 410
31.2 RTCEx Firmware driver defines . . . 415
31.2.1 RTCEx . . . 415
32 HAL SMARTCARD Generic Driver . . . 421
32.1 SMARTCARD Firmware driver registers structures . . . 421
32.1.1 SMARTCARD_InitTypeDef . . . 421
32.1.2 SMARTCARD_AdvFeatureInitTypeDef . . . 422
32.1.3 __SMARTCARD_HandleTypeDef . . . 423
32.2 SMARTCARD Firmware driver API description . . . 425
32.2.1 How to use this driver . . . 425
32.2.2 Callback registration . . . 426
32.2.3 Initialization and Configuration functions . . . 427
32.2.4 IO operation functions . . . 428
32.2.5 Peripheral State and Errors functions . . . 429
32.2.6 Detailed description of functions . . . 429
32.3 SMARTCARD Firmware driver defines . . . 439
32.3.1 SMARTCARD . . . 439
33 HAL SMARTCARD Extension Driver . . . 452

33.1 SMARTCARDEx Firmware driver API description . . . 452
33.1.1 SMARTCARD peripheral extended features . . . 452
33.1.2 Peripheral Control functions . . . 452
33.1.3 IO operation functions . . . 452
33.1.4 Peripheral FIFO Control functions . . . 452
33.1.5 Detailed description of functions . . . 453
33.2 SMARTCARDEx Firmware driver defines . . . 456
33.2.1 SMARTCARDEx . . . 456
34 HAL SMBUS Generic Driver . . . 461
34.1 SMBUS Firmware driver registers structures . . . 461
34.1.1 SMBUS_InitTypeDef . . . 461
34.1.2 __SMBUS_HandleTypeDef . . . 462
34.2 SMBUS Firmware driver API description . . . 463
34.2.1 How to use this driver . . . 463
34.2.2 Initialization and de-initialization functions . . . 465
34.2.3 IO operation functions . . . 466
34.2.4 Peripheral State and Errors functions . . . 466
34.2.5 Detailed description of functions . . . 467
34.3 SMBUS Firmware driver defines . . . 476
34.3.1 SMBUS . . . 476
35 HAL SMBUS Extension Driver . . . 484
35.1 SMBUSEx Firmware driver API description . . . 484
35.1.1 SMBUS peripheral Extended features . . . 484
35.1.2 How to use this driver . . . 484
35.1.3 WakeUp Mode Functions . . . 484
35.1.4 Fast Mode Plus Functions . . . 484
35.1.5 Detailed description of functions . . . 484
35.2 SMBUSEx Firmware driver defines . . . 485
35.2.1 SMBUSEx . . . 486
36 HAL SPI Generic Driver . . . 487
36.1 SPI Firmware driver registers structures . . . 487
36.1.1 SPI_InitTypeDef . . . 487
36.1.2 __SPI_HandleTypeDef . . . 488
36.2 SPI Firmware driver API description . . . 489
36.2.1 How to use this driver . . . 489
36.2.2 Initialization and de-initialization functions . . . 491
36.2.3 IO operation functions . . . 492

36.2.4 Peripheral State and Errors functions . . . 493
36.2.5 Detailed description of functions . . . 493
36.3 SPI Firmware driver defines . . . 502
36.3.1 SPI . . . 502
37 HAL SPI Extension Driver . . . 509
37.1 SPIEx Firmware driver API description . . . 509
37.1.1 IO operation functions . . . 509
37.1.2 Detailed description of functions . . . 509
38 HAL TIM Generic Driver . . . 510
38.1 TIM Firmware driver registers structures . . . 510
38.1.1 TIM_Base_InitTypeDef . . . 510
38.1.2 TIM_OC_InitTypeDef . . . 510
38.1.3 TIM_OnePulse_InitTypeDef . . . 511
38.1.4 TIM_IC_InitTypeDef . . . 512
38.1.5 TIM_Encoder_InitTypeDef . . . 512
38.1.6 TIM_ClockConfigTypeDef . . . 513
38.1.7 TIM_ClearInputConfigTypeDef . . . 513
38.1.8 TIM_MasterConfigTypeDef . . . 514
38.1.9 TIM_SlaveConfigTypeDef . . . 514
38.1.10 TIM_BreakDeadTimeConfigTypeDef . . . 514
38.1.11 __TIM_HandleTypeDef . . . 515
38.2 TIM Firmware driver API description . . . 517
38.2.1 TIMER Generic features . . . 517
38.2.2 How to use this driver . . . 518
38.2.3 Time Base functions . . . 519
38.2.4 TIM Output Compare functions . . . 520
38.2.5 TIM PWM functions . . . 520
38.2.6 TIM Input Capture functions . . . 521
38.2.7 TIM One Pulse functions . . . 521
38.2.8 TIM Encoder functions . . . 522
38.2.9 TIM Callbacks functions . . . 522
38.2.10 Detailed description of functions . . . 523
38.3 TIM Firmware driver defines . . . 562
38.3.1 TIM . . . 562
39 HAL TIM Extension Driver . . . 590
39.1 TIMEx Firmware driver registers structures . . . 590
39.1.1 TIM_HallSensor_InitTypeDef . . . 590

39.1.2 TIMEx_BreakInputConfigTypeDef . . . 590
39.2 TIMEx Firmware driver API description . . . 590
39.2.1 TIMER Extended features . . . 590
39.2.2 How to use this driver . . . 591
39.2.3 Timer Hall Sensor functions . . . 591
39.2.4 Timer Complementary Output Compare functions . . . 592
39.2.5 Timer Complementary PWM functions . . . 592
39.2.6 Timer Complementary One Pulse functions . . . 592
39.2.7 Peripheral Control functions . . . 592
39.2.8 Extended Callbacks functions . . . 593
39.2.9 Extended Peripheral State functions . . . 593
39.2.10 Detailed description of functions . . . 593
39.3 TIMEx Firmware driver defines . . . 609
39.3.1 TIMEx . . . 609
40 HAL UART Generic Driver . . . 612
40.1 UART Firmware driver registers structures . . . 612
40.1.1 UART_InitTypeDef . . . 612
40.1.2 UART_AdvFeatureInitTypeDef . . . 613
40.1.3 __UART_HandleTypeDef . . . 613
40.2 UART Firmware driver API description . . . 616
40.2.1 How to use this driver . . . 616
40.2.2 Callback registration . . . 616
40.2.3 Initialization and Configuration functions . . . 617
40.2.4 IO operation functions . . . 618
40.2.5 Peripheral Control functions . . . 619
40.2.6 Peripheral State and Error functions . . . 619
40.2.7 Detailed description of functions . . . 620
40.3 UART Firmware driver defines . . . 637
40.3.1 UART . . . 637
41 HAL UART Extension Driver . . . 658
41.1 UARTEx Firmware driver registers structures . . . 658
41.1.1 UART_WakeUpTypeDef . . . 658
41.2 UARTEx Firmware driver API description . . . 658
41.2.1 UART peripheral extended features . . . 658
41.2.2 Initialization and Configuration functions . . . 658
41.2.3 IO operation functions . . . 659
41.2.4 Peripheral Control functions . . . 659

41.2.5 Detailed description of functions . . . 660
41.3 UARTEx Firmware driver defines . . . 666
41.3.1 UARTEx . . . 667
42 HAL USART Generic Driver . . . 669
42.1 USART Firmware driver registers structures . . . 669
42.1.1 USART_InitTypeDef . . . 669
42.1.2 __USART_HandleTypeDef . . . 669
42.2 USART Firmware driver API description . . . 671
42.2.1 How to use this driver . . . 671
42.2.2 Callback registration . . . 672
42.2.3 Initialization and Configuration functions . . . 673
42.2.4 IO operation functions . . . 673
42.2.5 Peripheral State and Error functions . . . 675
42.2.6 Detailed description of functions . . . 675
42.3 USART Firmware driver defines . . . 686
42.3.1 USART . . . 686
43 HAL USART Extension Driver . . . 698
43.1 USARTEx Firmware driver API description . . . 698
43.1.1 USART peripheral extended features . . . 698
43.1.2 IO operation functions . . . 698
43.1.3 Peripheral Control functions . . . 698
43.1.4 Detailed description of functions . . . 698
43.2 USARTEx Firmware driver defines . . . 701
43.2.1 USARTEx . . . 701
44 HAL WWDG Generic Driver . . . 704
44.1 WWDG Firmware driver registers structures . . . 704
44.1.1 WWDG_InitTypeDef . . . 704
44.1.2 __WWDG_HandleTypeDef . . . 704
44.2 WWDG Firmware driver API description . . . 704
44.2.1 WWDG Specific features . . . 704
44.2.2 How to use this driver . . . 705
44.2.3 Initialization and Configuration functions . . . 706
44.2.4 IO operation functions . . . 706
44.2.5 Detailed description of functions . . . 706
44.3 WWDG Firmware driver defines . . . 708
44.3.1 WWDG . . . 708
45 LL ADC Generic Driver . . . 712

45.1 ADC Firmware driver registers structures . . . 712
45.1.1 LL_ADC_CommonInitTypeDef . . . 712
45.1.2 LL_ADC_InitTypeDef . . . 712
45.1.3 LL_ADC_REG_InitTypeDef . . . 712
45.2 ADC Firmware driver API description . . . 713
45.2.1 Detailed description of functions . . . 713
45.3 ADC Firmware driver defines . . . 788
45.3.1 ADC . . . 789
46 LL BUS Generic Driver . . . 814
46.1 BUS Firmware driver API description . . . 814
46.1.1 Detailed description of functions . . . 814
46.2 BUS Firmware driver defines . . . 832
46.2.1 BUS . . . 832
47 LL CORTEX Generic Driver . . . 835
47.1 CORTEX Firmware driver API description . . . 835
47.1.1 Detailed description of functions . . . 835
47.2 CORTEX Firmware driver defines . . . 842
47.2.1 CORTEX . . . 842
48 LL CRC Generic Driver . . . 846
48.1 CRC Firmware driver API description . . . 846
48.1.1 Detailed description of functions . . . 846
48.2 CRC Firmware driver defines . . . 853
48.2.1 CRC . . . 853
49 LL DMAMUX Generic Driver . . . 855
49.1 DMAMUX Firmware driver API description . . . 855
49.1.1 Detailed description of functions . . . 855
49.2 DMAMUX Firmware driver defines . . . 881
49.2.1 DMAMUX . . . 881
50 LL DMA Generic Driver . . . 890
50.1 DMA Firmware driver registers structures . . . 890
50.1.1 LL_DMA_InitTypeDef . . . 890
50.2 DMA Firmware driver API description . . . 891
50.2.1 Detailed description of functions . . . 891
50.3 DMA Firmware driver defines . . . 937
50.3.1 DMA . . . 937
51 LL EXTI Generic Driver . . . 944

51.1 EXTI Firmware driver registers structures . . . 944
51.1.1 LL_EXTI_InitTypeDef . . . 944
51.2 EXTI Firmware driver API description . . . 944
51.2.1 Detailed description of functions . . . 944
51.3 EXTI Firmware driver defines . . . 966
51.3.1 EXTI . . . 966
52 LL GPIO Generic Driver . . . 970
52.1 GPIO Firmware driver registers structures . . . 970
52.1.1 LL_GPIO_InitTypeDef . . . 970
52.2 GPIO Firmware driver API description . . . 970
52.2.1 Detailed description of functions . . . 970
52.3 GPIO Firmware driver defines . . . 990
52.3.1 GPIO . . . 990
53 LL I2C Generic Driver . . . 994
53.1 I2C Firmware driver registers structures . . . 994
53.1.1 LL_I2C_InitTypeDef . . . 994
53.2 I2C Firmware driver API description . . . 994
53.2.1 Detailed description of functions . . . 994
53.3 I2C Firmware driver defines . . . 1043
53.3.1 I2C . . . 1043
54 LL IWDG Generic Driver . . . 1050
54.1 IWDG Firmware driver API description . . . 1050
54.1.1 Detailed description of functions . . . 1050
54.2 IWDG Firmware driver defines . . . 1054
54.2.1 IWDG . . . 1054
55 LL PWR Generic Driver . . . 1056
55.1 PWR Firmware driver API description . . . 1056
55.1.1 Detailed description of functions . . . 1056
55.2 PWR Firmware driver defines . . . 1075
55.2.1 PWR . . . 1075
56 LL RCC Generic Driver . . . 1078
56.1 RCC Firmware driver registers structures . . . 1078
56.1.1 LL_RCC_ClocksTypeDef . . . 1078
56.2 RCC Firmware driver API description . . . 1078
56.2.1 Detailed description of functions . . . 1078
56.3 RCC Firmware driver defines . . . 1111

56.3.1 RCC . . . 1111
57 LL RTC Generic Driver . . . 1122
57.1 RTC Firmware driver registers structures . . . 1122
57.1.1 LL_RTC_InitTypeDef . . . 1122
57.1.2 LL_RTC_TimeTypeDef . . . 1122
57.1.3 LL_RTC_DateTypeDef . . . 1122
57.1.4 LL_RTC_AlarmTypeDef . . . 1123
57.2 RTC Firmware driver API description . . . 1123
57.2.1 Detailed description of functions . . . 1123
57.3 RTC Firmware driver defines . . . 1174
57.3.1 RTC . . . 1174
58 LL SPI Generic Driver . . . 1182
58.1 SPI Firmware driver registers structures . . . 1182
58.1.1 LL_SPI_InitTypeDef . . . 1182
58.1.2 LL_I2S_InitTypeDef . . . 1183
58.2 SPI Firmware driver API description . . . 1183
58.2.1 Detailed description of functions . . . 1183
58.3 SPI Firmware driver defines . . . 1226
58.3.1 SPI . . . 1227
59 LL SYSTEM Generic Driver . . . 1232
59.1 SYSTEM Firmware driver API description . . . 1232
59.1.1 Detailed description of functions . . . 1232
59.2 SYSTEM Firmware driver defines . . . 1255
59.2.1 SYSTEM . . . 1255
60 LL TIM Generic Driver . . . 1260
60.1 TIM Firmware driver registers structures . . . 1260
60.1.1 LL_TIM_InitTypeDef . . . 1260
60.1.2 LL_TIM_OC_InitTypeDef . . . 1260
60.1.3 LL_TIM_IC_InitTypeDef . . . 1261
60.1.4 LL_TIM_ENCODER_InitTypeDef . . . 1261
60.1.5 LL_TIM_HALLSENSOR_InitTypeDef . . . 1262
60.1.6 LL_TIM_BDTR_InitTypeDef . . . 1263
60.2 TIM Firmware driver API description . . . 1264
60.2.1 Detailed description of functions . . . 1265
60.3 TIM Firmware driver defines . . . 1357
60.3.1 TIM . . . 1357
61 LL USART Generic Driver . . . 1378

61.1 USART Firmware driver registers structures 1378
61.1.1 LL_USART_InitTypeDef 1378
61.1.2 LL_USART_ClockInitTypeDef 1378
61.2 USART Firmware driver API description 1379
61.2.1 Detailed description of functions 1379
61.3 USART Firmware driver defines 1475
61.3.1 USART 1475
62 LL UTILS Generic Driver 1485
62.1 UTILS Firmware driver API description 1485
62.1.1 System Configuration functions 1485
62.1.2 Detailed description of functions 1485
62.2 UTILS Firmware driver defines 1488
62.2.1 UTILS 1488
63 LL WWDG Generic Driver 1489
63.1 WWDG Firmware driver API description 1489
63.1.1 Detailed description of functions 1489
63.2 WWDG Firmware driver defines 1493
63.2.1 WWDG 1493
64 FAQs 1495
Revision history 1498
List of tables 1516
List of figures 1517

List of tables

Table 1. Acronyms and definitions . . . 3
Table 2. HAL driver files . . . 6
Table 3. User-application files . . . 6
Table 4. API classification . . . 11
Table 5. List of devices supported by HAL/LL drivers . . . 12
Table 6. HAL API naming rules . . . 14
Table 7. Macros handling interrupts and specific clock configurations . . . 15
Table 8. Callback functions . . . 16
Table 9. HAL generic APIs . . . 17
Table 10. HAL extension APIs . . . 17
Table 11. Define statements used for HAL configuration . . . 21
Table 12. Description of GPIO_InitTypeDef structure . . . 22
Table 13. Description of EXTI configuration macros . . . 24
Table 14. MSP functions . . . 28
Table 15. Timeout values . . . 31
Table 16. LL driver files . . . 35
Table 17. Common peripheral initialization functions . . . 37
Table 18. Optional peripheral initialization functions . . . 38
Table 19. Specific interrupt, DMA request and status flags management . . . 39
Table 20. Available function formats . . . 39
Table 21. Peripheral clock activation/deactivation management . . . 39
Table 22. Peripheral activation/deactivation management . . . 39
Table 23. Peripheral configuration management . . . 40
Table 24. Peripheral register management . . . 40
Table 25. Document revision history . . . 1498

List of figures

Figure 1. Example of project template 8
Figure 2. Adding device-specific functions 18
Figure 3. Adding family-specific functions 18
Figure 4. Adding new peripherals 19
Figure 5. File inclusion model. 19
Figure 6. HAL driver model 26
Figure 7. Low-layer driver folders 36
Figure 8. Low-layer driver CMSIS files 37

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2025 STMicroelectronics – All rights reserved