UM3363

User manual

Description of STM32WB0 HAL and LL 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 STM32CubeWB0 for STM32WB0 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 RTOS, FAT file system, STM32_BLE including Bluetooth® LE, services and stack, certified with Bluetooth® SIG.
  - All embedded software utilities, delivered with a full set of examples.

The HAL driver layer provides a simple, generic multi-instance set of APIs (application programming interfaces) to interact with the upper layer (application, libraries, and stacks). The HAL driver APIs are split into two categories: generic APIs, which provide common and generic functions for all the STM32 series and extension APIs, which include specific and customized functions for a given line or part number. The HAL drivers include a complete set of ready-to-use APIs that simplify the user application implementation. For example, the communication peripherals contain APIs to initialize and configure the peripheral, manage data transfers in polling mode, handle interrupts or DMA, and manage communication errors. The HAL drivers are feature-oriented instead of 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 fully documented.

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 STM32CubeWB0 MCU Package runs on STM32WB0 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
AES	Advanced encryption standard
ANSI	American national standards institute
API	Application programming interface
BSP	Board support package
CAN	Controller area network
CEC	Consumer electronic controller
CMSIS	Cortex microcontroller software interface standard
COMP	Comparator
CPU	Central processing unit
CRC	CRC calculation unit
CSS	Clock security system
DAC	Digital-to-analog converter
DMA	Direct memory access
DMAMUX	Direct memory access request multiplexer
DTS	Digital temperature sensor
EXTI	External interrupt/event controller
FLASH	Flash memory
FW	Firewall
GPIO	General-purpose I/Os
HAL	Hardware abstraction layer
HCD	USB host controller driver
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	Infrared data association
IWDG	Independent watchdog
JPEG	Joint photographic experts group
LPTIM	Low-power timer
LPUART	Low-power universal asynchronous receiver/transmitter
MCO	Microcontroller clock output
MMC	MultiMediaCard
MPU	Memory protection unit
MSP	MCU specific package
NAND	NAND flash memory
NOR	NOR flash memory
NVIC	Nested vectored interrupt controller
OTG-FS	USB on-the-go full-speed
PKA	Public key accelerator

Acronym	Definition
PCD	USB peripheral controller driver
PPP	STM32 peripheral or block
PWR	Power controller
RAMECC	RAM ECC monitoring
RCC	Reset and clock controller
RNG	Random number generator
RTC	Real-time clock
SD	Secure digital
SDMMC	SD/SDIO/MultiMediaCard card host interface
SMARTCARD	Smartcard IC
SMBUS	System management bus
SPI	Serial peripheral interface
SRAM	SRAM external memory
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
USB	Universal serial bus
VREFBUF	Voltage reference buffer

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
stm32wb0_hal_ppp.c	Main peripheral/module driver file It includes the APIs that are common to all STM32 devices. *Example: stm32wb0_hal_adc.c, stm32wb0_hal_irda.c.*
stm32wb0_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:stm32wb0_hal_adc.h,stm32wb0_hal_irda.h.*
stm32wb0_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:stm32wb0_hal_adc_ex.c,stm32wb0_hal_flash_ex.c.*
stm32wb0_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: stm32wb0_hal_adc_ex.h,stm32wb0_hal_flash_ex.h.*
stm32wb0_hal.c	This file is used for HAL initialization and contains DBGMCU, remap and time delay based on SysTick APIs.
stm32wb0_hal.h	stm32wb0_hal.c header file
stm32wb0_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.
stm32wb0_hal_conf_template.h	Template file allowing the drivers to be customized for a given application
stm32wb0_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_stm32wb0.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_stm32wb0.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.
stm32wb0_flash.icf (optional)	Linker file for EWARM toolchain allowing mainly the adaptation of the stack/heap size to fit the application requirements.
stm32wb0_hal_msp.c	This file contains the MSP initialization and deinitialization (main routine and callbacks) of the peripheral used in the user application.
stm32wb0_hal_conf.h	This file allows the HAL drivers to be customized for a specific application.

File	Description
	It is not mandatory to modify this configuration. The application can use the default configuration without any modification.
stm32wb0_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 *stm32wb0*_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.

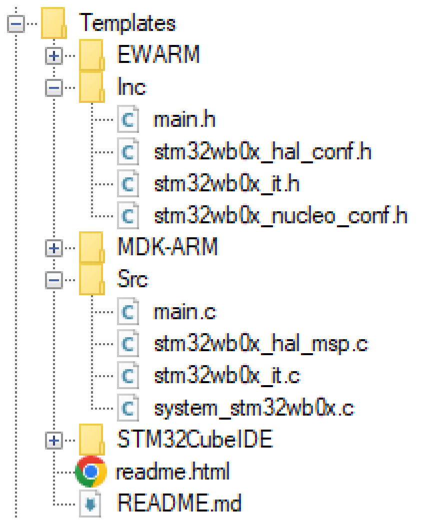
The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

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 a
chieve 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 APIs contains **family-specific APIs** that apply to a given family. They are located in the extension HAL driver file (see example below related to the ADC).
  –
  ```
  HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t Si
  ngleDiff); uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32
  _t SingleDiff);
  ```

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	STM32WB09xx	STM32WB07xx	STM32WB06xx	STM32WB05xx
stm32wb0x_hal.c	yes	yes	yes	yes
stm32wb0x_hal_adc.c	yes	yes	yes	yes
stm32wb0x_hal_adc_ex.c	yes	yes	yes	yes
stm32wb0x_hal_cortex.c	yes	yes	yes	yes
stm32wb0x_hal_crc.c	yes	yes	yes	yes
stm32wb0x_hal_crc_ex.c	yes	yes	yes	yes
stm32wb0x_hal_dma.c	yes	yes	yes	yes
stm32wb0x_hal_flash.c	yes	yes	yes	yes
stm32wb0x_hal_flash_ex.c	yes	yes	yes	yes
stm32wb0x_hal_gpio.c	yes	yes	yes	yes
stm32wb0x_hal_i2c.c	yes	yes	yes	yes
stm32wb0x_hal_i2c_ex.c	yes	yes	yes	yes
stm32wb0x_hal_i2s.c	yes	yes	yes	yes
stm32wb0x_hal_irda.c	yes	yes	yes	yes
stm32wb0x_hal_iwdg.c	yes	yes	yes	yes
stm32wb0x_hal_pka.c	yes	yes	yes	yes
stm32wb0x_hal_pwr.c	yes	yes	yes	yes
stm32wb0x_hal_pwr_ex.c	yes	yes	yes	yes
stm32wb0x_hal_radio.c	yes	yes	yes	yes
stm32wb0x_hal_radio_timer.c	yes	yes	yes	yes
stm32wb0x_hal_rcc.c	yes	yes	yes	yes
stm32wb0x_hal_rcc_ex.c	yes	yes	yes	yes
stm32wb0x_hal_rng.c	yes	yes	yes	yes
stm32wb0x_hal_rtc.c	yes	yes	yes	yes
stm32wb0x_hal_rtc_ex.c	yes	yes	yes	yes
stm32wb0x_hal_smartcard.c	yes	yes	yes	yes
stm32wb0x_hal_smartcard_ex.c	yes	yes	yes	yes
stm32wb0x_hal_smbus.c	yes	yes	yes	yes
stm32wb0x_hal_smbus_ex.c	yes	yes	yes	yes
stm32wb0x_hal_spi.c	yes	yes	yes	yes
stm32wb0x_hal_spi_ex.c	yes	yes	yes	yes
stm32wb0x_hal_tim.c	yes	yes	yes	yes
stm32wb0x_hal_tim_ex.c	yes	yes	yes	yes
stm32wb0x_hal_uart.c	yes	yes	yes	yes
stm32wb0x_hal_uart_ex.c	yes	yes	yes	yes
stm32wb0x_hal_usart.c	yes	yes	yes	yes
stm32wb0x_hal_usart_ex.c	yes	yes	yes	yes
stm32wb0x_ll_adc.c	yes	yes	yes	yes

Peripheral/Module	STM32WB09xx	STM32WB07xx	STM32WB06xx	STM32WB05xx
stm32wb0x_ll_crc.c	yes	yes	yes	yes
stm32wb0x_ll_dma.c	yes	yes	yes	yes
stm32wb0x_ll_gpio.c	yes	yes	yes	yes
stm32wb0x_ll_i2c.c	yes	yes	yes	yes
stm32wb0x_ll_i2s.c	yes	yes	yes	yes
stm32wb0x_ll_lpuart.c	yes	yes	yes	yes
stm32wb0x_ll_pka.c	yes	yes	yes	yes
stm32wb0x_ll_pwr.c	yes	yes	yes	yes
stm32wb0x_ll_rcc.c	yes	yes	yes	yes
stm32wb0x_ll_rng.c	yes	yes	yes	yes
stm32wb0x_ll_rtc.c	yes	yes	yes	yes
stm32wb0x_ll_spi.c	yes	yes	yes	yes
stm32wb0x_ll_system.c	yes	yes	yes	yes
stm32wb0x_ll_tim.c	yes	yes	yes	yes
stm32wb0x_ll_usart.c	yes	yes	yes	yes
stm32wb0x_ll_utils.c	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	*stm32wb0_hal_ppp (c/h)*	*stm32wb0_hal_ppp_ex (c/h)*	*stm32wb0_ 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 STM32WB0 reference manuals.
- Peripheral registers are declared in the PPP_TypeDef structure (for example ADC_TypeDef) in stm32wb0x.h header file: stm32wb0xxx.h corresponds to stm32wb05.h, stm32wb06.h, stm32wb07.h, and stm32wb09.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 SYSTICK are two Arm® Cortex® core features. The APIs related to these features are located in the stm32wb0_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 stm32wb0_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, stm32wb0_hal_ppp_ex.c, that includes all the specific functions and define statements (stm32wb0_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

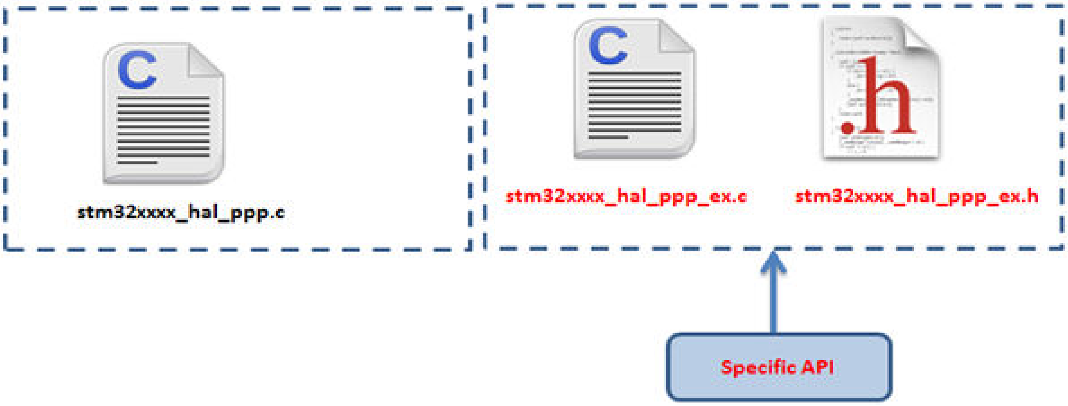
3.7.2 HAL extension model cases

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 stm32wb0_hal_ppp_ex.c extension file. They are named HAL_PPPEx_Function().

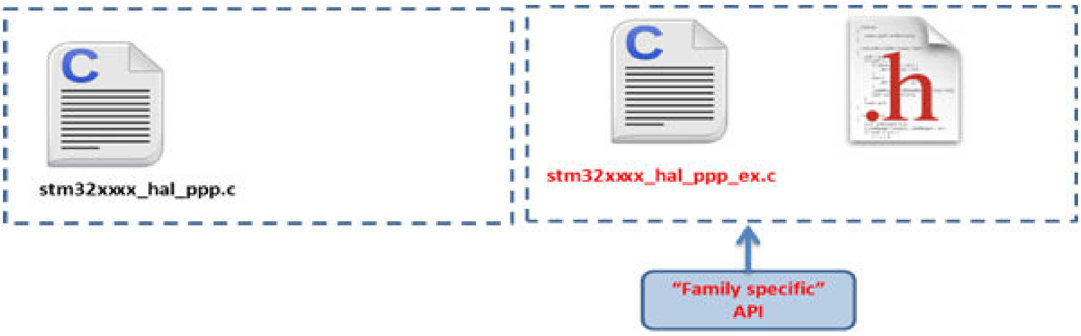
Figure 2. Adding device-specific functions



Adding a family-specific function

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

Figure 3. Adding family-specific functions

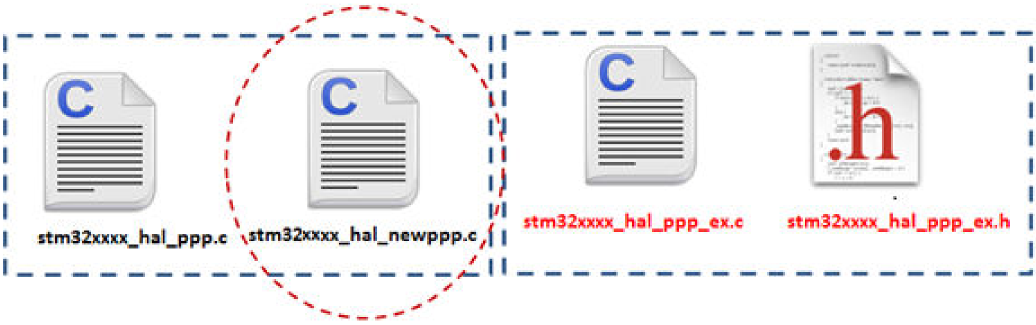


Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module (newPPP) are added in a new stm32wb0_hal_newppp.c. However, the inclusion of this file is selected in the stm32wb0_hal_conf.h using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4. Adding new peripherals



Example: stm32wb0_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.

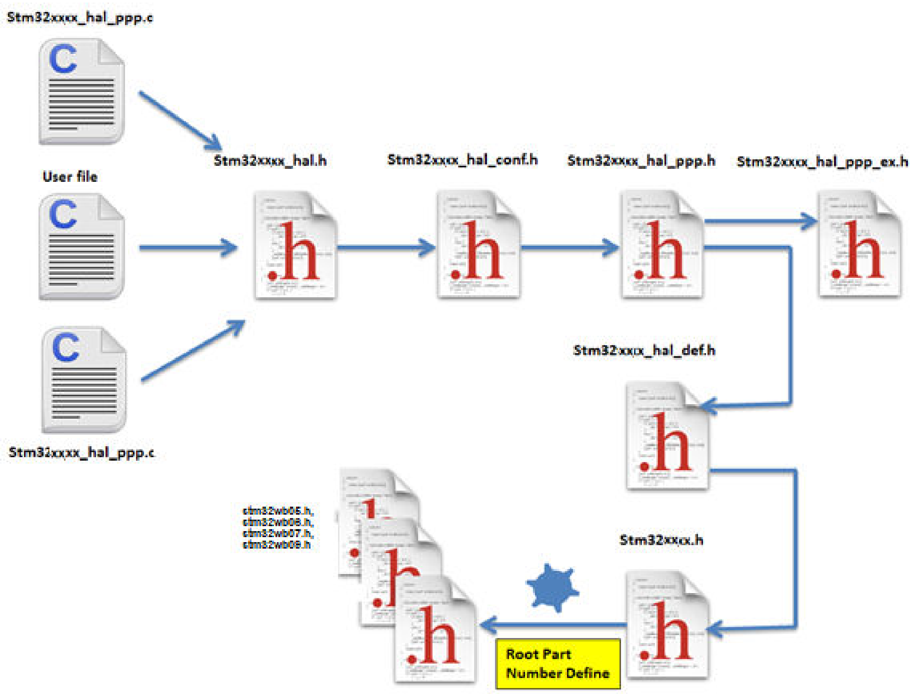
Example:

```
#if defined (STM32WB09xx)
typedef struct
{
(…)
}PPP_InitTypeDef;
#endif
```

3.8 File inclusion model

The header of the common HAL driver file (stm32wb0_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 stm32wb0_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 *stm32wb0_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 stm32wb0_hal_def.h file calls the stm32wb0.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, stm32wb0_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.	32 000 000 Hz
HSE_STARTUP_TIMEOUT	Timeout for HSE startup, expressed in ms	100 ms
HSI_VALUE	Value of the internal oscillator (HSI) expressed in Hz.	64 000 000 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

Configuration item	Description	Default Value
LSE_STARTUP_TIMEOUT	Timeout for LSE startup, expressed in ms	5000 ms
VDD_VALUE	VDD value	3300 mV
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
RC64MPLL_VALUE	Value of the Internal oscillator in Hz	64 000 000 Hz
LSE_DRIVE_LEVEL	Drive level for LSE oscillator	0
CFG_HW_RCC_HSE_CAPACITOR_TUNE	HSE capacitor tuning	32

Note: *The stm32wb0_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 stm32wb0_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, PLL).
- HAL_RCC_ClockConfig (RCC_ClkInitTypeDef *RCC_ClkInitStruct, uint32_t FLatency). This function
  - Selects the system clock source
  - Configures SYSCLK 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 and RF). In this case, the clock configuration is performed by an extended API defined in stm32wb0_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, PCLK1 or PCLK2)
- MCO and CSS configuration functions

A set of macros are defined in stm32wb0_hal_rcc.h and stm32wb0_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 stm32wb0_it.c and implement HAL_GPIO_EXTI_Callback().

The table below describes the GPIO_InitTypeDef structure field.

Table 12. Description of GPIO_InitTypeDef structure

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

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, stm32wb0_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:

- PVD configuration, enabling/disabling, and interrupt handling
  - HAL_PWR_ConfigPVD()
  - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
  - HAL_PWR_PVD_IRQHandler()
  - HAL_PWR_PVDCallback()
- Wake-up pin configuration
  - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
  - HAL_PWR_GetClearWakeupSource()
- Low-power mode entry
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterDEEPSTOPMode()
- STM32WB0 low-power configuration features:
  - HAL_PWR_ConfigDEEPSTOP()
  - HAL_PWR_ConfigSHUTDOWN()
- STM32WB0 low-power management features:
  - HAL_PWREx_EnterSHUTDOWNMode()
- Sleep-on-exit and SEV on pending configuration functions:
  - HAL_PWR_EnableSleepOnExit(void)
  - HAL_PWR_DisableSleepOnExit(void)
  - HAL_PWR_EnableSEVOnPend(void)
  - HAL_PWR_DisableSEVOnPend(void);
- Interrupt handler functions:
  - HAL_PWR_WKUP_IRQHandler(void)
  - HAL_PWR_WKUPx_Callback(uint32_t WakeupIOs)

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 PVD, RTC, USB, and Ethernet, 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 stm32wb0_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 and normal control modes
- 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”).*

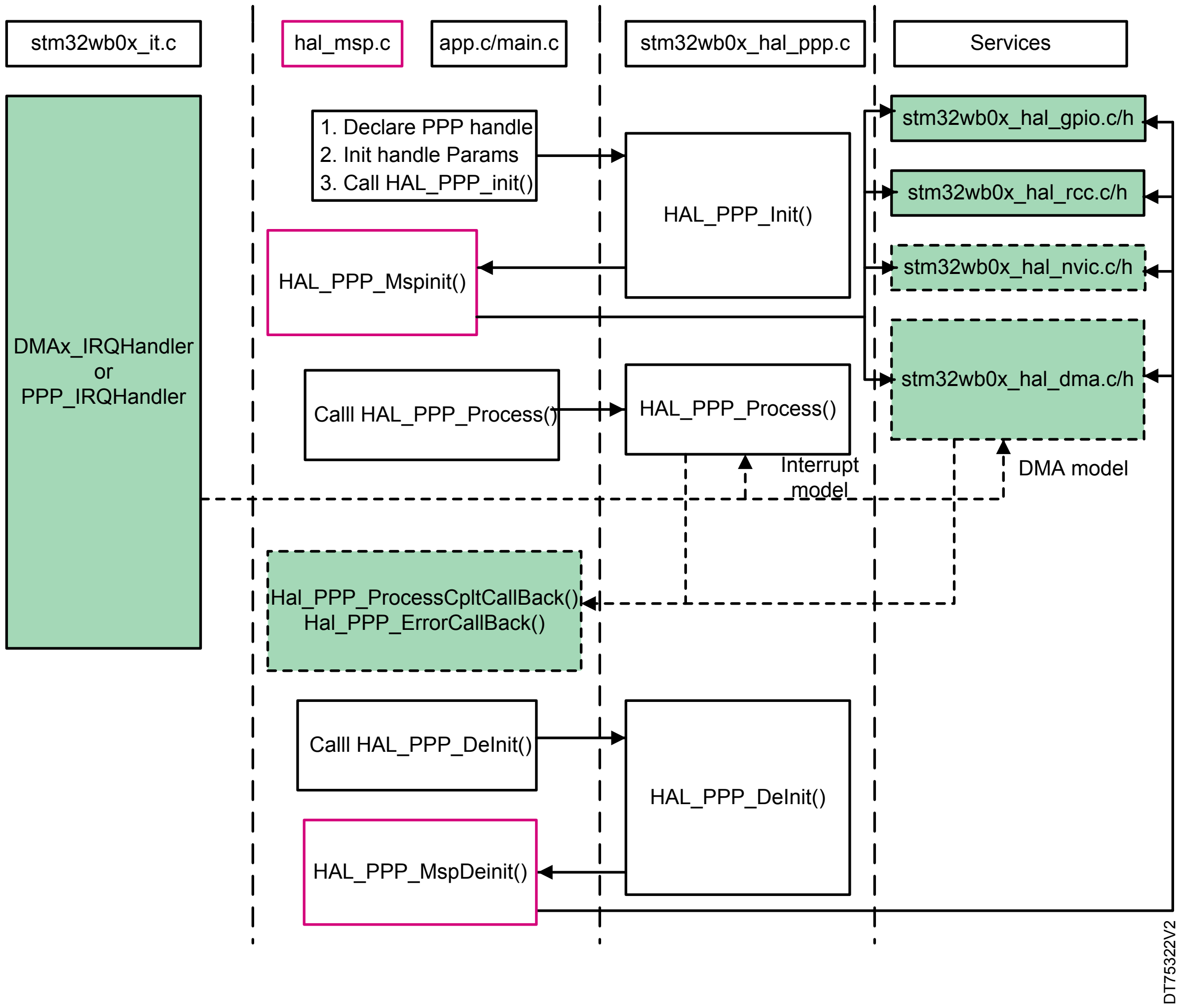
Note: *DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However, when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.*

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 stm32wb0_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 deinitalizations.
- 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 64 MHz based on the HSE clock and PLL:

```
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
/** Configure the SYSCLKSource and SYSCLKDivider
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
Error_Handler();
}
/** Configure the SYSCLKSource clocks dividers */
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_RC64MPLL;
RCC_ClkInitStruct.SYSCLKDivider = RCC_RC64MPLL_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_WAIT_STATES_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 *stm32wb0_hal_msp.c* file in the user folders. An *stm32wb0_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.

stm32wb0_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 *stm32wb0_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)
{
}
```

stm32wb0_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 *stm32wb0_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)
{
}
```

stm32wb0_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 stm32wb0_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 *stm32wb0_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 stm32wb0_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.

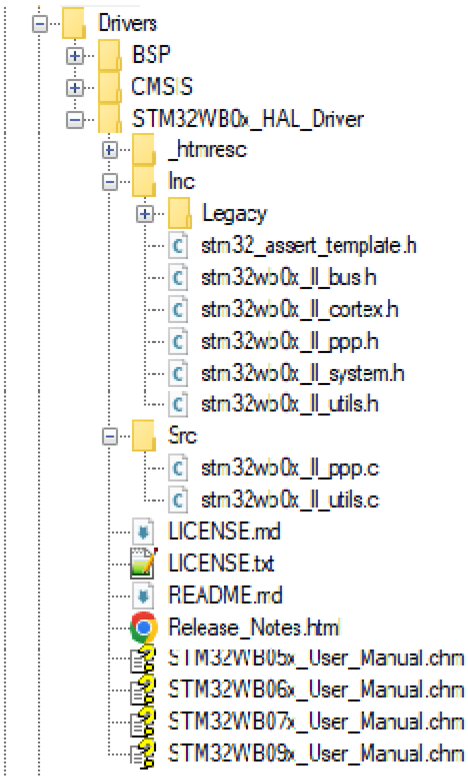
Table 16. LL driver files

File	Description
stm32wb0_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB1_GRP1_EnableClock*
stm32wb0_ll_ppp.h/.c	stm32wb0_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 stm32wb0_ll_ppp.h file. The low-layer PPP driver is a standalone module. To use it, the application must include it in the stm32wb0_ll_ppp.h file.
stm32wb0_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")
stm32wb0_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.
stm32wb0_ll_system.h	System-related operations. Example: LL_SYSCFG_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.

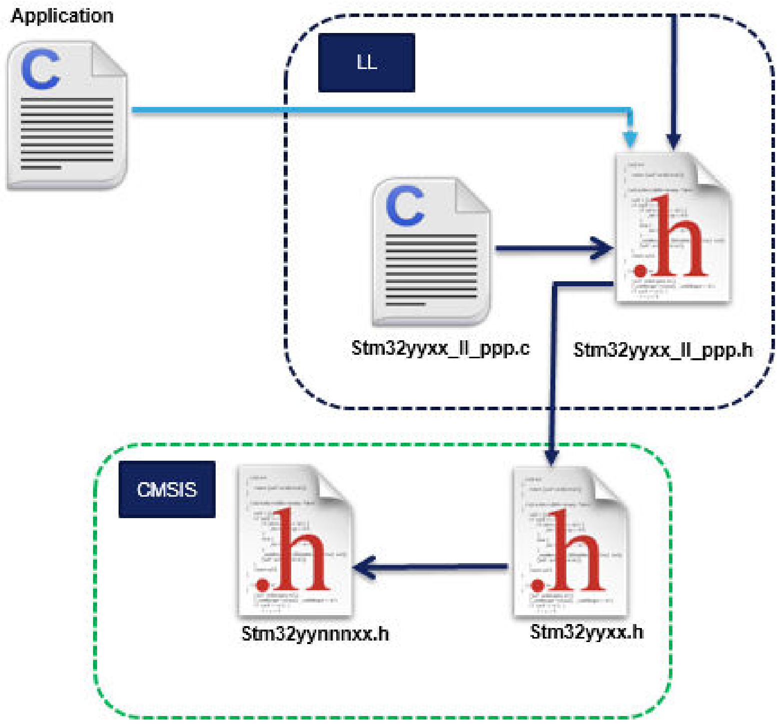
Figure 7. Low-layer driver folders



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 stm32wb0_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)

Functions	Return type	Parameters	Examples
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*

Item	Action	Format
Interrupts	Enable	*LL_PPP_EnableIT_BITNAME*
	Disable	*LL_PPP_DisableIT_BITNAME*
	Get	*LL_PPP_IsEnabledIT_BITNAME*
DMA	Enable	*LL_PPP_EnableDMAReq_BITNAME*
	Disable	*LL_PPP_DisableDMAReq_BITNAME*
	Get	*LL_PPP_IsEnabledDMAReq_BITNAME*

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

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

Table 21. Peripheral clock activation/deactivation management

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

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

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

Table 22. Peripheral activation/deactivation management

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

- ***Peripheral configuration management:*** Set/get a peripheral configuration 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 stm32wb0_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 STM32CubeWB0 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 stm32wb0 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 1 ms 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 revision identifier
- Get the device identifier
- Get the unique device 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_GetVERID()*
- *HAL_GetDEVID()*
- *HAL_GetJTAGID()*
- *HAL_GetUIDw0()*
- *HAL_GetUIDw1()*
- *HAL_GetUIDw2()*

6.1.4 Detailed description of functions

HAL_Init

Function name

HAL_StatusTypeDef HAL_Init (void)

Function description

This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it configures the SysTick to generate an interrupt each 1 ms, clocked by the system clock.

Return values

- **HAL:** status

Notes

- SysTick is used as time base for the HAL_Delay() function, the application need to ensure that the SysTick time base is always set to 1 ms to have correct HAL operation.

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 1 ms 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 1 ms 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:** period in Hz

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_GetVERID

Function name

uint32_t HAL_GetVERID (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 device identifier.

Return values

- **Device:** identifier

HAL_GetJTAGID

Function name

uint32_t HAL_GetJTAGID (void)

Function description

Returns the Jtag device identifier.

Return values

- **Device:** identifier

HAL_GetUIDw0

Function name

uint32_t HAL_GetUIDw0 (void)

Function description

Return the first word of the unique device identifier (UID based on 64 bits)

Return values

- **Device:** identifier

HAL_GetUIDw1

Function name

uint32_t HAL_GetUIDw1 (void)

Function description

Return the second word of the unique device identifier (UID based on 64 bits)

Return values

- **Device:** identifier

HAL_GetUIDw2

Function name

uint32_t HAL_GetUIDw2 (void)

Function description

6.2 HAL Firmware driver defines

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

6.2.1 HAL

HAL

HAL Exported Macros

__HAL_SYSCFG_REMAPMEMORY_FLASH

__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:
  - SYSCFG_BOOT_MAINFLASH
  - SYSCFG_BOOT_SRAM

__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 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 0x00000000

SYSCFG_BOOT_SRAM

SRAM1 mapped at 0x00000000

Fast-mode Plus on GPIO

SYSCFG_FASTMODEPLUS_PA0

Enable Fast-mode Plus on PA0

SYSCFG_FASTMODEPLUS_PA1

Enable Fast Mode Plus on PA1

7 HAL ADC Generic Driver

7.1 ADC Firmware driver registers structures

7.1.1 ADC_AnalogWDGConfTypeDef

ADC_AnalogWDGConfTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t LowThreshold***
- ***uint32_t HighThreshold***
- ***uint32_t ChannelMask***
- ***uint32_t EventType***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
  Set the lower threshold for the watchdog. This parameter is a 12-bit value.
- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
  Set the higher threshold for the watchdog. This parameter is a 12-bit value.
- ***uint32_t ADC_AnalogWDGConfTypeDef::ChannelMask***
  Select the input channels the watchdog has to use. This parameter can be a value of ***ADC_AWD_CHANNEL***.
- ***uint32_t ADC_AnalogWDGConfTypeDef::EventType***
  Select the event type used for the watchdog. This parameter can be a value of ***ADC_AWD_EVENTTYPE***.

7.1.2 ADC_CalibrationPointTypeDef

ADC_CalibrationPointTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t Number***
- ***uint32_t Offset***
- ***uint32_t Gain***

Field Documentation

- ***uint32_t ADC_CalibrationPointTypeDef::Number***
  Select the calibration point number This parameter can be a value of ***ADC_HAL_CALIB_POINT*** Caution: Only one offset is allowed per input mode (Single positive, single negative and differential) and voltage range. This parameter overwrites the last setting. Note: Use the ADC_CALIB_NONE to skip calibration parameter for this channel and leave offset and gain untouched.
- ***uint32_t ADC_CalibrationPointTypeDef::Offset***
  Define the offset to be subtracted from the raw converted data. Offset value must be a positive number. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF.
- ***uint32_t ADC_CalibrationPointTypeDef::Gain***
  Specify the ADC gain compensation coefficient to be applied to ADC raw conversion data, based on following formula: DATA = DATA(raw) * (gain compensation coef) / 4096 This parameter value can be:
  – value "0": Gain will be set to default parameters
  – value in range [0x001; 0xFFF]: Gain compensation will be set to specified value

7.1.3 ADC_ChannelConfTypeDef

ADC_ChannelConfTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t VoltRange***
- ***ADC_CalibrationPointTypeDef CalibrationPoint***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
  Specify the channel type the channel number code for the 1st conversion of the sequence. This parameter can be a value of ***ADC_HAL_CH_NUM_CODE***.
- ***uint32_t ADC_ChannelConfTypeDef::Rank***
  Assign a specific sequence number to the channel. This parameter can be a value of ***ADC_HAL_EC_SEQ_RANKS***.
- ***uint32_t ADC_ChannelConfTypeDef::VoltRange***
  Specify the voltage range for the selected channel. This parameter can be a value of ***ADC_HAL_INPUT_VOLTAGE_RANGE***.
- ***ADC_CalibrationPointTypeDef ADC_ChannelConfTypeDef::CalibrationPoint***
  Specify the calibration parameter for the selected channel.

7.1.4 ADC_DSConfTypeDef

ADC_DSConfTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t DataWidth***
- ***uint32_t DataRatio***

Field Documentation

- ***uint32_t ADC_DSConfTypeDef::DataWidth***
  Specify the width of the output data from the Down Sampler (DS). This parameter can be a value of ***ADC_HAL_DS_DATA_WIDTH***.
- ***uint32_t ADC_DSConfTypeDef::DataRatio***
  Specify the ratio of the output data from the Down Sampler (DS). This parameter can be a value of ***ADC_HAL_DS_RATIO***.

7.1.5 ADC_DFConfTypeDef

ADC_DFConfTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t DynamicMode***
- ***uint32_t HighPassFilter***
- ***uint32_t OutputDataFormatMode***
- ***uint32_t InputDataFormatMode***
- ***uint32_t FractionalInterpolator***
- ***uint32_t CICFilterDecimatorHalfFactor***
- ***uint32_t MCICOutputFrequency***

Field Documentation

- ***uint32_t ADC_DFConfTypeDef::DynamicMode***
  Specify the input dynamic range for the Decimation Filter (DF). This parameter can be a value of ADC_HAL_DF_DYN_RANGE.
- ***uint32_t ADC_DFConfTypeDef::HighPassFilter***
  Specify whether the high pass filter feature is enabled or disabled. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_DFConfTypeDef::OutputDataFormatMode***
  Specify the signed/unsigned output data format. This parameter can be a value of ADC_HAL_DF_DATA_FORMAT.
- ***uint32_t ADC_DFConfTypeDef::InputDataFormatMode***
  Specify the signed/unsigned input data format. This parameter can be a value of ADC_HAL_DF_DATA_FORMAT.
- ***uint32_t ADC_DFConfTypeDef::FractionalInterpolator***
  Specify whether fractional interpolator is enabled or disabled. This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_DFConfTypeDef::CICFilterDecimatorHalfFactor***
  Specify if the decimator factor of the CIC filter is using integer or half. Note: this bit must be set only for the generation of a data rate at 44.1 kps from ADC data at 1 MHz. Unpredictable result may happen if set for other configuration. This parameter can be a value of ADC_HAL_CIC_DECIMATOR_HALF_FACTOR.

- ***uint32_t ADC_DFConfTypeDef::MCICOutputFrequency***
  Specify the output frequency of the CIC filter. This parameter can be a value of ADC_HAL_MICROPHONE_OUT_FREQ.

7.1.6 ADC_PDMConfTypeDef

ADC_PDMConfTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t ClockDivider***

Field Documentation

- ***uint32_t ADC_PDMConfTypeDef::ClockDivider***
  Specify the clock divider for the (PDM) Interface. This parameter can be a value of ADC_HAL_PDM_CLK_DIVIDER.

7.1.7 ADC_PGAConfTypeDef

ADC_PGAConfTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t Bias***
- ***uint32_t Gain***

Field Documentation

- ***uint32_t ADC_PGAConfTypeDef::Bias***
  Specify the bias voltage (offset) for the Programmable Gain Amplifier (PGA). This parameter can be a value of ADC_HAL_PGA_BIAS.
- ***uint32_t ADC_PGAConfTypeDef::Gain***
  Specify the gain for the Programmable Gain Amplifier (PGA). This parameter can be a value of ADC_HAL_PGA_GAIN.

7.1.8 ADC_InitTypeDef

ADC_InitTypeDef is defined in the stm32wb0x_hal_adc.h

Data Fields

- ***uint32_t ConversionType***
- ***FunctionalState ContinuousConvMode***
- ***uint32_t SequenceLength***
- ***uint32_t SamplingMode***
- ***uint32_t SampleRate***
- ***ADC_DSConfTypeDef DownSamplerConfig***
- ***uint32_t InvertOutputMode***
- ***uint32_t Overrun***

Field Documentation

- ***uint32_t ADC_InitTypeDef::ConversionType***
  Specifies the conversion type: through the Down Sampler (DS) or through the Decimation Filter (DF). This parameter can be a value of ***ADC_HAL_CONVERSION_TYPE***.
- ***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::SequenceLength***
  Specify the length of the conversion sequence. This parameter must be a number between Min_Data = 1 and Max_Data = 16.
- ***uint32_t ADC_InitTypeDef::SamplingMode***
  Specifies the input sampling mode. This parameter can be a value of ***ADC_HAL_SAMPLING_METHOD***.
- ***uint32_t ADC_InitTypeDef::SampleRate***
  Specify the ADC sample rate. This parameter can be a value of ***ADC_HAL_SAMPLE_RATE***.

- ***ADC_DSConfTypeDef ADC_InitTypeDef::DownSamplerConfig***
  Specifies the Down Sampler parameters.
- ***uint32_t ADC_InitTypeDef::InvertOutputMode***
  Specifies the bit to bit inversion (1' complement) for differential input and/or single negative input. This parameter can be a value of ***ADC_HAL_DATA_INVERT***.
- ***uint32_t ADC_InitTypeDef::Overrun***
  Specifies the overrung policy applied to the data. This parameter can be a value of ***ADC_HAL_OVERRUN_CONFIG***.

7.1.9 __ADC_HandleTypeDef

__ADC_HandleTypeDef is defined in the stm32wb0x_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***
- ***void(* ConvCpltCallback***
- ***void(* ConvHalfCpltCallback***
- ***void(* LevelOutOfWindowCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***ADC_TypeDef* __ADC_HandleTypeDef::Instance***
  Register base address
- ***ADC_InitTypeDef __ADC_HandleTypeDef::Init***
  Parameters for ADC initialization
- ***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
- ***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::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

- Conversion frequency is up to 1 Msps.
- Three input voltage ranges are supported (0 V : 1.2 V, 0 V : 2.4 V, 0 V : 3.6 V).
- Up to eight analog single ended channels or four analog differential inputs or a mix of both.
- Temperature sensor conversion.
- Battery level conversion up to 3.6 V.
- Continuous or single acquisition.
- ADC Down Sampler for multi-purpose applications to improve analog performance while off-loading the CPU (ratio adjustable from 1 to 128).
- A watchdog feature to inform when data is outside thresholds.
- DMA capability.
- Interrupt sources with flags.

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.
   - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
     - __HAL_RCC_ADC_CLK_ENABLE(); (mandatory)
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 ADC_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, channels parameters

1. Configure the ADC parameters using function HAL_ADC_Init().
2. Configure the channels using the function HAL_ADC_ConfigChannel().
3. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AWDConfig().

Execution of ADC conversions

1. 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()*
- *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).
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 callback
- ErrorCallback : ADC error 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 callback
- ErrorCallback : ADC error 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 Analog Timing Control functions

This section provides functions allowing to:

- Configure the synchronisation between SMPS and ADC start of conversion
- Configure the ADC PGA VBias precharge duration
- Configure the ADC LDO delay

This section contains the following APIs:

- *HAL_ADC_SMPSSyncEnable()*
- *HAL_ADC_SMPSSyncDisable()*
- *HAL_ADC_LDODelayConfig()*
- *HAL_ADC_VBiasPrechargeDelayConfig()*

7.2.4 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on a conversion sequence
- Configure the analog watchdog

This section contains the following APIs:

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

7.2.5 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()*

7.2.6 Detailed description of functions

HAL_ADC_Init

Function name

HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)

Function description

Initialize the ADC peripheral 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".

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

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 complete callback ID
  – HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC watchdog callback ID
  – HAL_ADC_ERROR_CB_ID ADC error callback ID
  – HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  – HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit 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 complete callback ID
  – HAL_ADC_LEVEL_OUT_OF_WINDOW_1_CB_ID ADC watchdog callback ID
  – HAL_ADC_ERROR_CB_ID ADC error callback ID
  – HAL_ADC_MSPINIT_CB_ID ADC Msp Init callback ID
  – HAL_ADC_MSPDEINIT_CB_ID ADC Msp DeInit 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.

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 and disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

Notes

- : ADC peripheral disable is forcing stop of potential conversion.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for a 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.
- This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion. 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.

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_IRQ_FLAG_OVRFL ADC decimation filter saturated event (1) ADC_IRQ_FLAG_OVRDF ADC decimation filter overrun event (1) ADC_IRQ_FLAG_EODF ADC decimation filter conversion completed event (1) ADC_IRQ_FLAG_OVRDS ADC Down Sampler overrun event ADC_IRQ_FLAG_AWD1 ADC analog watchdog event ADC_IRQ_FLAG_EOS ADC sequence of conversion completed event ADC_IRQ_FLAG_EODS ADC Down Sampler conversion completed Event (1) todo: Only available on STM32WB0x
- **Timeout:** Timeout value in millisecond.

Return values

- **HAL:** status

Notes

- The relevant flag is cleared if found to be set, except for ADC_IRQ_FLAG_OVRDS. Indeed, the latter is reset only if hadc->Init.Overrun field is set to ADC_NEW_DATA_IS_KEPT. 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_IRQ_FLAG_OVRDS);

HAL_ADC_Start_IT

Function name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function description

Enable ADC and start conversion 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.

HAL_ADC_Stop_IT

Function name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion, disable interrution of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- **HAL:** status.

HAL_ADC_Start_DMA

Function name

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

Function description

Enable ADC, start conversion 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. The ADC continuous mode is enabled in this DMA acquisition mode.

HAL_ADC_Stop_DMA

Function name

HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

Function description

Stop ADC conversion, 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 conversion result (raw value).

Parameters

- **hadc:** ADC handle

Return values

- **ADC:** conversion data

Notes

- 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. The flag EODS is rising when the Down Sampler conversion is completed. To clear those flags, either use function: in programming model IT: HAL_ADC_IRQHandler(), in programming model polling: HAL_ADC_PollForConversion() or __HAL_ADC_CLEAR_FLAG(&hadc, ADC_IRQ_FLAG_EOS).

HAL_ADC_GetValue_DF

Function name

uint32_t HAL_ADC_GetValue_DF (const ADC_HandleTypeDef * hadc)

Function description

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 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_ConfigChannel

Function name

HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, const ADC_ChannelConfTypeDef * sConfigChannel)

Function description

Configure a channel to be assigned to ADC conversion sequence.

Parameters

- **hadc:** ADC handle
- **sConfigChannel:** Structure of ADC channel configuration and position in ADC conversion sequence.

Return values

- **HAL:** status

HAL_ADC_DFConfig

Function name

HAL_StatusTypeDef HAL_ADC_DFConfig (ADC_HandleTypeDef * hadc, const ADC_DFConfTypeDef * ConfigDF)

Function description

HAL_ADC_AnalogWDGConfig

Function name

HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, const ADC_AnalogWDGConfTypeDef * ConfigWatchdog)

Function description

Configure the watchdog

Parameters

- **hadc:** ADC handle
- **ConfigWatchdog:** Structure of ADC watchdog configuration

Return values

- **HAL:** status

Notes

- Possibility to update parameters on the fly: This function initializes the 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.

HAL_ADC_SetMicrophoneChannel

Function name

HAL_StatusTypeDef HAL_ADC_SetMicrophoneChannel (ADC_HandleTypeDef * hadc, uint32_t MicrophoneChannel)

Function description

HAL_ADC_PDMConfig

Function name

HAL_StatusTypeDef HAL_ADC_PDMConfig (ADC_HandleTypeDef * hadc, const ADC_PDMConfTypeDef * ConfigPDM)

Function description

HAL_ADC_PDMStart

Function name

HAL_StatusTypeDef HAL_ADC_PDMStart (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_PDMStop

Function name

HAL_StatusTypeDef HAL_ADC_PDMStop (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_PGAConfig

Function name

HAL_StatusTypeDef HAL_ADC_PGAConfig (ADC_HandleTypeDef * hadc, const ADC_PGAConfTypeDef * ConfigPGA)

Function description

HAL_ADC_SetOccasionalSource

Function name

HAL_StatusTypeDef HAL_ADC_SetOccasionalSource (ADC_HandleTypeDef * hadc, uint32_t Source)

Function description

HAL_ADC_StartOccasionalMode

Function name

HAL_StatusTypeDef HAL_ADC_StartOccasionalMode (ADC_HandleTypeDef * hadc)

Function description

HAL_ADC_SMPSSyncEnable

Function name

HAL_StatusTypeDef HAL_ADC_SMPSSyncEnable (ADC_HandleTypeDef * hadc)

Function description

Enable the synchronization of the ADC start conversion with a pulse generated by the SMPS.

Parameters

- **hadc:** ADC handle

Notes

- The synchronization should only be enabled when using the SMPS with an ADC clock different from 32 MHz.

HAL_ADC_SMPSSyncDisable

Function name

HAL_StatusTypeDef HAL_ADC_SMPSSyncDisable (ADC_HandleTypeDef * hadc)

Function description

Disable the synchronization of the ADC start conversion with a pulse generated by the SMPS.

Parameters

- **hadc:** ADC handle

Notes

- The synchronization should only be enabled when using the SMPS with an ADC clock different from 32 MHz.

HAL_ADC_VBiasPrechargeDelayConfig

Function name

HAL_StatusTypeDef HAL_ADC_VBiasPrechargeDelayConfig (ADC_HandleTypeDef * hadc, uint32_t Delay_us)

Function description

HAL_ADC_LDODelayConfig

Function name

HAL_StatusTypeDef HAL_ADC_LDODelayConfig (ADC_HandleTypeDef * hadc, uint32_t Delay_us)

Function description

Configure the LDO stabilization time.

Parameters

- **hadc:** ADC handle
- **Delay_us:** delay in us. This parameter must be a number between Min_Data = 4 and Max_Data = 1020.

Notes

- The delay in microseconds corresponding to the duration of a waiting time to be inserted between the ADC_LDO enable and the ADC ON to let the LDO stabilize before starting a conversion.
- The delay value should be an increment of 4us. Otherwise the function configure an approximate available value close to the input value.
- This value is configured at reset to a default value. This value can be reconfigured using ADC_DEFAULT_LDO_DELAY_US.

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(hadc) & HAL_ADC_STATE_DS_BUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc) & 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)

7.3 ADC Firmware driver defines

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

7.3.1 ADC

ADC

ADC watchdog channel selection for bit mask definitions

ADC_AWD_CH_VINM0

ADC watchdog channel selection: VINM0 to ADC negative input

ADC_AWD_CH_VINM1

ADC watchdog channel selection: VINM1 to ADC negative input

ADC_AWD_CH_VINM2

ADC watchdog channel selection: VINM2 to ADC negative input

ADC_AWD_CH_VINM3

ADC watchdog channel selection: VINM3 to ADC negative input

ADC_AWD_CH_VINP0

ADC watchdog channel selection: VINP0 to ADC positive input

ADC_AWD_CH_VINP1

ADC watchdog channel selection: VINP1 to ADC positive input

ADC_AWD_CH_VINP2

ADC watchdog channel selection: VINP2 to ADC positive input

ADC_AWD_CH_VINP3

ADC watchdog channel selection: VINP3 to ADC positive input

ADC_AWD_CH_MICROM

ADC watchdog channel selection: MICROM to ADC negative input

ADC_AWD_CH_MICROP

ADC watchdog channel selection: MICROP to ADC positive input

ADC_AWD_CH_VBAT

ADC watchdog channel selection: VBAT to ADC negative input

ADC_AWD_CH_TEMPSENSOR

ADC watchdog channel selection: TEMPSENSOR to ADC positive input

ADC_AWD_CH_GND_NEG

ADC watchdog channel selection: GND to ADC negative input

ADC_AWD_CH_GND_POS

ADC watchdog channel selection: GND to ADC positive input

ADC_AWD_CH_VDDA_NEG

ADC watchdog channel selection: VDDA to ADC negative input

ADC_AWD_CH_VDDA_POS

ADC watchdog channel selection: VDDA to ADC positive input

ADC watchdog event type definitions

ADC_AWD_EVENT_POLLING

ADC watchdog event type polling

ADC_AWD_EVENT_INTERRUPT

ADC watchdog event type interrupt

ADC Error Code

HAL_ADC_ERROR_NONE

No error

HAL_ADC_ERROR_INTERNAL

Internal error

HAL_ADC_ERROR_OVR

Data overrun error

HAL_ADC_ERROR_DMA

DMA transfer error

HAL_ADC_ERROR_INVALID_CALLBACK

Invalid Callback error

ADC Exported Constants

ADC_DEFAULT_LDO_DELAY_US

Default delay for LDO stabilization (uS)

ADC_DEFAULT_VBIAS_PRECH_DELAY_US

Default delay for the VBIAS precharge pulse duration (uS)

ADC calibration points definitions

ADC_CALIB_POINT_1

ADC calibration point 1

ADC_CALIB_POINT_2

ADC calibration point 2

ADC_CALIB_POINT_3

ADC calibration point 3

ADC_CALIB_POINT_4

ADC calibration point 4

ADC_CALIB_NONE

ADC calibration point disabled: correction are left untouched for the selected channel

ADC channel number code for conversion definitions

ADC_CHANNEL_VINM0

ADC channel for VINM0 to single negative input

ADC_CHANNEL_VINM1

ADC channel for VINM1 to single negative input

ADC_CHANNEL_VINM2

ADC channel for VINM2 to single negative input

ADC_CHANNEL_VINM3

ADC channel for VINM3 to single negative input

ADC_CHANNEL_VINP0

ADC channel for VINP0 to single positive input

ADC_CHANNEL_VINP1

ADC channel for VINP1 to single positive input

ADC_CHANNEL_VINP2

ADC channel for VINP2 to single positive input

ADC_CHANNEL_VINP3

ADC channel for VINP3 to single positive input

ADC_CHANNEL_VINP0_VINM0

ADC channel for VINP0 - VINM0 to differential input

ADC_CHANNEL_VINP1_VINM1

ADC channel for VINP1 - VINM1 to differential input

ADC_CHANNEL_VINP2_VINM2

ADC channel for VINP2 - VINM2 to differential input

ADC_CHANNEL_VINP3_VINM3

ADC channel for VINP3 - VINM3 to differential input

ADC_CHANNEL_VBAT

ADC channel for VBAT, battery level detector

ADC_CHANNEL_TEMPSENSOR

ADC channel for temperature sensor

ADC conversion type definitions

ADC_CONVERSION_WITH_DS

ADC conversion through the Down Sampler (DS)

ADC_CONVERSION_WITH_DF

ADC conversion through the Decimation Filter (DF)

ADC invert bit to bit data output for differential or

ADC_DATA_INVERT_NONE

No inversion

ADC_DATA_INVERT_DIFF

Inversion only for differential input

ADC_DATA_INVERT_SING

Inversion only for single negative input

ADC_DATA_INVERT_BOTH

Inversion for both input type

ADC Down Sampler data width definitions

ADC_DS_DATA_WIDTH_12_BIT

ADC Down Sampler data width 12 bits

ADC_DS_DATA_WIDTH_13_BIT

ADC Down Sampler data width 13 bits

ADC_DS_DATA_WIDTH_14_BIT

ADC Down Sampler data width 14 bits

ADC_DS_DATA_WIDTH_15_BIT

ADC Down Sampler data width 15 bits

ADC_DS_DATA_WIDTH_16_BIT

ADC Down Sampler data width 16 bits

ADC Down Sampler ratio definitions

ADC_DS_RATIO_1

ADC Down Sampler ratio 1, no down sampling (default)

ADC_DS_RATIO_2

ADC Down Sampler ratio 2

ADC_DS_RATIO_4

ADC Down Sampler ratio 4

ADC_DS_RATIO_8

ADC Down Sampler ratio 8

ADC_DS_RATIO_16

ADC Down Sampler ratio 16

ADC_DS_RATIO_32

ADC Down Sampler ratio 32

ADC_DS_RATIO_64

ADC Down Sampler ratio 64

ADC_DS_RATIO_128

ADC Down Sampler ratio 128

ADC Sequencer ranks

ADC_RANK_1

ADC sequencer rank 1

ADC_RANK_2

ADC sequencer rank 2

ADC_RANK_3

ADC sequencer rank 3

ADC_RANK_4

ADC sequencer rank 4

ADC_RANK_5

ADC sequencer rank 5

ADC_RANK_6

ADC sequencer rank 6

ADC_RANK_7

ADC sequencer rank 7

ADC_RANK_8

ADC sequencer rank 8

ADC_RANK_9

ADC sequencer rank 9

ADC_RANK_10

ADC sequencer rank 10

ADC_RANK_11

ADC sequencer rank 11

ADC_RANK_12

ADC sequencer rank 12

ADC_RANK_13

ADC sequencer rank 13

ADC_RANK_14

ADC sequencer rank 14

ADC_RANK_15

ADC sequencer rank 15

ADC_RANK_16

ADC sequencer rank 16

HAL ADC macro to manage HAL ADC handle,

__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 a combination of the following values:
  - ADC_IRQ_EN_OVRFL ADC Decimation Filter saturated interrupt source
  - ADC_IRQ_EN_OVRDF ADC Decimation Filter overrung interrupt source
  - ADC_IRQ_EN_OVRDS ADC Down Sampler overrun interrupt source
  - ADC_IRQ_EN_AWD1 ADC Watchdog alert interrupt source
  - ADC_IRQ_EN_EOC ADC End of conversion interrupt source
  - ADC_IRQ_EN_EOS ADC End of conversion of a sequence interrupt source
  - ADC_IRQ_EN_EODF ADC End of conversion from Decimation Filter interrupt source
  - ADC_IRQ_EN_EODS ADC End of conversion from Down Sampler interrupt source

Return value:

- None

__HAL_ADC_DISABLE_IT

Description:

- Disable ADC interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be a combination of the following values:
  - ADC_IRQ_EN_OVRFL ADC Decimation Filter saturated interrupt source
  - ADC_IRQ_EN_OVRDF ADC Decimation Filter overrung interrupt source
  - ADC_IRQ_EN_OVRDS ADC Down Sampler overrun interrupt source
  - ADC_IRQ_EN_AWD1 ADC Watchdog alert interrupt source
  - ADC_IRQ_EN_EOC ADC End of conversion interrupt source
  - ADC_IRQ_EN_EOS ADC End of conversion of a sequence interrupt source
  - ADC_IRQ_EN_EODF ADC End of conversion from Decimation Filter interrupt source
  - ADC_IRQ_EN_EODS ADC End of conversion from Down Sampler interrupt source

Return value:

- None

__HAL_ADC_GET_IT_SOURCE

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be one of the following values:
  - ADC_IRQ_EN_OVRFL ADC Decimation Filter saturated interrupt source
  - ADC_IRQ_EN_OVRDF ADC Decimation Filter overrung interrupt source
  - ADC_IRQ_EN_OVRDS ADC Down Sampler overrun interrupt source
  - ADC_IRQ_EN_AWD1 ADC Watchdog alert interrupt source
  - ADC_IRQ_EN_EOC ADC End of conversion interrupt source
  - ADC_IRQ_EN_EOS ADC End of conversion of a sequence interrupt source
  - ADC_IRQ_EN_EODF ADC End of conversion from Decimation Filter interrupt source
  - ADC_IRQ_EN_EODS ADC End of conversion from Down Sampler interrupt source

Return value:

- State: of interruption (SET or RESET)

__HAL_ADC_GET_FLAG

Description:

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

Parameters:

- __HANDLE__: ADC handle
- __FLAG__: ADC flag This parameter can be one of the following values:
  - ADC_IRQ_FLAG_OVRFL ADC Decimation Filter saturated interrupt source
  - ADC_IRQ_FLAG_OVRDF ADC Decimation Filter overrung interrupt source
  - ADC_IRQ_FLAG_OVRDS ADC Down Sampler overrun interrupt source
  - ADC_IRQ_FLAG_AWD1 ADC Watchdog alert interrupt source
  - ADC_IRQ_FLAG_EOC ADC End of conversion interrupt source
  - ADC_IRQ_FLAG_EOS ADC End of conversion of a sequence interrupt source
  - ADC_IRQ_FLAG_EODF ADC End of conversion from Decimation Filter interrupt source
  - ADC_IRQ_FLAG_EODS ADC End of conversion from Down Sampler interrupt source

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 a combination of the following values:
  - ADC_IRQ_FLAG_OVRFL ADC Decimation Filter saturated interrupt source
  - ADC_IRQ_FLAG_OVRDF ADC Decimation Filter overrung interrupt source
  - ADC_IRQ_FLAG_OVRDS ADC Down Sampler overrun interrupt source
  - ADC_IRQ_FLAG_AWD1 ADC Watchdog alert interrupt source
  - ADC_IRQ_FLAG_EOC ADC End of conversion interrupt source
  - ADC_IRQ_FLAG_EOS ADC End of conversion of a sequence interrupt source
  - ADC_IRQ_FLAG_EODF ADC End of conversion from Decimation Filter interrupt source
  - ADC_IRQ_FLAG_EODS ADC End of conversion from Down Sampler interrupt source

Return value:

- None

HAL ADC helper macro

__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_VINM0
  - ADC_CHANNEL_VINM1
  - ADC_CHANNEL_VINM2
  - ADC_CHANNEL_VINM3
  - ADC_CHANNEL_VINP0
  - ADC_CHANNEL_VINP1
  - ADC_CHANNEL_VINP2
  - ADC_CHANNEL_VINP3
  - ADC_CHANNEL_VINP0_VINM0
  - ADC_CHANNEL_VINP1_VINM1
  - ADC_CHANNEL_VINP2_VINM2
  - ADC_CHANNEL_VINP3_VINM3
  - ADC_CHANNEL_VBAT
  - ADC_CHANNEL_TEMPSENSOR

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.

__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_WIDTH_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - ADC_DS_DATA_WIDTH_12_BIT
  - ADC_DS_DATA_WIDTH_13_BIT
  - ADC_DS_DATA_WIDTH_14_BIT
  - ADC_DS_DATA_WIDTH_15_BIT
  - ADC_DS_DATA_WIDTH_16_BIT
- __ADC_WIDTH_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - ADC_DS_DATA_WIDTH_12_BIT
  - ADC_DS_DATA_WIDTH_13_BIT
  - ADC_DS_DATA_WIDTH_14_BIT
  - ADC_DS_DATA_WIDTH_15_BIT
  - ADC_DS_DATA_WIDTH_16_BIT

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:

- __INPUT_VOLTAGE_RANGE__: Analog reference voltage (unit: mV)
  - ADC_VIN_RANGE_3V6
  - ADC_VIN_RANGE_2V4
  - ADC_VIN_RANGE_1V2
- __ADC_DATA__: ADC conversion data (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  - ADC_DS_DATA_WIDTH_12_BIT
  - ADC_DS_DATA_WIDTH_13_BIT
  - ADC_DS_DATA_WIDTH_14_BIT
  - ADC_DS_DATA_WIDTH_15_BIT
  - ADC_DS_DATA_WIDTH_16_BIT

Return value:

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

__HAL_ADC_CALC_DIFF_DATA_TO_VOLTAGE

Description:

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

Parameters:

- __INPUT_VOLTAGE_RANGE__: Analog reference voltage (unit: mV)
  - ADC_VIN_RANGE_3V6
  - ADC_VIN_RANGE_2V4
  - ADC_VIN_RANGE_1V2
- __ADC_DATA__: ADC conversion data (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  - ADC_DS_DATA_WIDTH_12_BIT
  - ADC_DS_DATA_WIDTH_13_BIT
  - ADC_DS_DATA_WIDTH_14_BIT
  - ADC_DS_DATA_WIDTH_15_BIT
  - ADC_DS_DATA_WIDTH_16_BIT

Return value:

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

__HAL_ADC_CALC_TEMPERATURE

Description:

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

Parameters:

- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  – ADC_DS_DATA_WIDTH_12_BIT
  – ADC_DS_DATA_WIDTH_13_BIT
  – ADC_DS_DATA_WIDTH_14_BIT
  – ADC_DS_DATA_WIDTH_15_BIT
  – ADC_DS_DATA_WIDTH_16_BIT

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS().

__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_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  – ADC_DS_DATA_WIDTH_12_BIT
  – ADC_DS_DATA_WIDTH_13_BIT
  – ADC_DS_DATA_WIDTH_14_BIT
  – ADC_DS_DATA_WIDTH_15_BIT
  – ADC_DS_DATA_WIDTH_16_BIT

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet).

ADC input voltage range definitions

ADC_VIN_RANGE_1V2

ADC input voltage range up to 1.2 V

ADC_VIN_RANGE_2V4

ADC input voltage range up to 2.4 V

ADC_VIN_RANGE_3V6

ADC input voltage range up to 3.6 V

ADC interrupts enable definitions

ADC_IRQ_EN_OVRDS

ADC IRQ enable OVRDS

ADC_IRQ_EN_AWD1

ADC IRQ enable AWD1

ADC_IRQ_EN_EOS

ADC IRQ enable EOS

ADC_IRQ_EN_EOC

ADC IRQ enable EOC

ADC_IRQ_EN_EODS

ADC IRQ enable EODS

ADC IRQ_ENABLE register mask definitions

ADC_IRQ_EN_MASK

ADC IRQ_STATUS register mask definitions

ADC_IRQ_FLAG_OVRDS

ADC IRQ flag OVRDS Down Sampler overrun

ADC_IRQ_FLAG_AWD1

ADC IRQ flag AWD1 Analog watchdog event

ADC_IRQ_FLAG_EOS

ADC IRQ flag EOS End of conversion Sequence

ADC_IRQ_FLAG_EOC

ADC IRQ flag EOC End of conversion

ADC_IRQ_FLAG_EODS

ADC IRQ flag EODS End of Down Sampler conversion

ADC_IRQ_FLAGS_MASK

ADC operation modes definitions

ADC_OP_MODE_ADC

ADC operation mode ADC

ADC_OP_MODE_AUDIO

ADC operation mode AUDIO

ADC_OP_MODE_FULL

ADC operation Full mode

ADC overrun configuration definitions

ADC_NEW_DATA_IS_LOST

Previous data is preserved, new data is lost.

ADC_NEW_DATA_IS_KEPT

Previous data is overwritten, new data is kept.

ADC sample rate definitions

ADC_SAMPLE_RATE_16
ADC conversion rate at F_ADC_CLK / 16

ADC_SAMPLE_RATE_20
ADC conversion rate at F_ADC_CLK / 20

ADC_SAMPLE_RATE_24
ADC conversion rate at F_ADC_CLK / 24

ADC_SAMPLE_RATE_28
ADC conversion rate at F_ADC_CLK / 28

ADC_SAMPLE_RATE_32
ADC conversion rate at F_ADC_CLK / 32

ADC_SAMPLE_RATE_36
ADC conversion rate at F_ADC_CLK / 36

ADC_SAMPLE_RATE_40
ADC conversion rate at F_ADC_CLK / 40

ADC_SAMPLE_RATE_44
ADC conversion rate at F_ADC_CLK / 44

ADC_SAMPLE_RATE_48
ADC conversion rate at F_ADC_CLK / 48

ADC_SAMPLE_RATE_52
ADC conversion rate at F_ADC_CLK / 52

ADC_SAMPLE_RATE_56
ADC conversion rate at F_ADC_CLK / 56

ADC_SAMPLE_RATE_60
ADC conversion rate at F_ADC_CLK / 60

ADC_SAMPLE_RATE_64
ADC conversion rate at F_ADC_CLK / 64

ADC_SAMPLE_RATE_68
ADC conversion rate at F_ADC_CLK / 68

ADC_SAMPLE_RATE_72
ADC conversion rate at F_ADC_CLK / 72

ADC_SAMPLE_RATE_76
ADC conversion rate at F_ADC_CLK / 76

ADC_SAMPLE_RATE_80
ADC conversion rate at F_ADC_CLK / 80

ADC_SAMPLE_RATE_84
ADC conversion rate at F_ADC_CLK / 84

ADC_SAMPLE_RATE_88
ADC conversion rate at F_ADC_CLK / 88

ADC_SAMPLE_RATE_92

ADC conversion rate at F_ADC_CLK / 92

ADC_SAMPLE_RATE_96

ADC conversion rate at F_ADC_CLK / 96

ADC_SAMPLE_RATE_100

ADC conversion rate at F_ADC_CLK / 100

ADC_SAMPLE_RATE_104

ADC conversion rate at F_ADC_CLK / 104

ADC_SAMPLE_RATE_108

ADC conversion rate at F_ADC_CLK / 108

ADC_SAMPLE_RATE_112

ADC conversion rate at F_ADC_CLK / 112

ADC_SAMPLE_RATE_116

ADC conversion rate at F_ADC_CLK / 116

ADC_SAMPLE_RATE_120

ADC conversion rate at F_ADC_CLK / 120

ADC_SAMPLE_RATE_124

ADC conversion rate at F_ADC_CLK / 124

ADC_SAMPLE_RATE_128

ADC conversion rate at F_ADC_CLK / 128

ADC_SAMPLE_RATE_132

ADC conversion rate at F_ADC_CLK / 132

ADC_SAMPLE_RATE_136

ADC conversion rate at F_ADC_CLK / 136

ADC_SAMPLE_RATE_140

ADC conversion rate at F_ADC_CLK / 140

ADC input sampling method definitions

ADC_SAMPLING_AT_START

Sampling only at conversion start (default)

ADC_SAMPLING_AT_END

Sampling sampling phase starts after end of conversion, and stops upon trigger event (Also known as Bulb sampling mode).

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(hadc) & HAL_ADC_STATEBUSY) != 0UL) " " if ((HAL_ADC_GetState(hadc) & HAL_ADC_STATE_AWD1) != 0UL) " ADC not yet initialized or disabled

HAL_ADC_STATE_READY

ADC peripheral ready to use

HAL_ADC_STATE_BUSY_INTERNAL

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

HAL_ADC_STATE_TIMEOUT

TimeOut occurrence

HAL_ADC_STATE_ERROR_INTERNAL

Internal error occurrence

HAL_ADC_STATE_ERROR_CONFIG

Configuration error occurrence

HAL_ADC_STATE_ERROR_DMA

DMA error occurrence

HAL_ADC_STATE_DS_BUSY

A conversion on down sampler data path is ongoing or can occur

HAL_ADC_STATE_DS_EOC

Converted data available in downsampler data register

HAL_ADC_STATE_DS_OVR

Overrun on down sampler data path occurrence

HAL_ADC_STATE_DF_BUSY

A conversion on decimation filter path is ongoing or can occur

HAL_ADC_STATE_DF_EOC

Converted data available in decimation filter data register

HAL_ADC_STATE_DF_OVR

Overrun occurrence in decimation filter data

HAL_ADC_STATE_AWD1

Out-of-window occurrence of ADC analog watchdog

8 HAL CORTEX Generic Driver

8.1 CORTEX Firmware driver registers structures

8.1.1 MPU_Region_InitTypeDef

MPU_Region_InitTypeDef is defined in the stm32wb0x_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***

8.2 CORTEX Firmware driver API description

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

8.2.1 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.
  – Enables the SysTick Interrupt.
  – Starts the SysTick Counter.
- 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

8.2.2 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()*

8.2.3 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_IRQHandler()*
- *HAL_SYSTICK_Callback()*

- *HAL_MPU_Disable()*
- *HAL_MPU_Enable()*
- *HAL_MPU_ConfigRegion()*

8.2.4 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 stm32wb0x.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 stm32wb0x 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)

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)

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)

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)

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)

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)

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 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

Disables the MPU

Return values

- **None:**

HAL_MPU_ConfigRegion

Function name

void HAL_MPU_ConfigRegion (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:**

8.3 CORTEX Firmware driver defines

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

8.3.1 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 IRQ NVIC Priority

NVIC_CRITICAL_PRIORITY

Critical Priority

NVIC_HIGH_PRIORITY

High Priority

NVIC_MED_PRIORITY

Medium Priority

NVIC_LOW_PRIORITY

Low Priority

9 HAL CRC Generic Driver

9.1 CRC Firmware driver registers structures

9.1.1 CRC_InitTypeDef

CRC_InitTypeDef is defined in the stm32wb0x_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

9.1.2 CRC_HandleTypeDef

CRC_HandleTypeDef is defined in the stm32wb0x_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

9.2 CRC Firmware driver API description

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

9.2.1 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

9.2.2 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()*

9.2.3 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()*

9.2.4 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()*

9.2.5 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

9.3 CRC Firmware driver defines

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

9.3.1 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

10 HAL CRC Extension Driver

10.1 CRCEx Firmware driver API description

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

10.1.1 How to use this driver

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

10.1.2 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()*

10.1.3 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

10.2 CRCEx Firmware driver defines

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

10.2.1 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

11 HAL DMA Generic Driver

11.1 DMA Firmware driver registers structures

11.1.1 DMA_InitTypeDef

DMA_InitTypeDef is defined in the stm32wb0x_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***

11.1.2 __DMA_HandleTypeDef

__DMA_HandleTypeDef is defined in the stm32wb0x_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*
- *DMA_TypeDef * DmaBaseAddress*
- *uint32_t ChannelIndex*
- *DMAMUX_Channel_TypeDef * DMAmuxChannel*

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
- ***DMA_TypeDef* __DMA_HandleTypeDef::DmaBaseAddress***
  DMA Channel Base Address
- ***uint32_t __DMA_HandleTypeDef::ChannelIndex***
  DMA Channel Index
- ***DMAMUX_Channel_TypeDef* __DMA_HandleTypeDef::DMAmuxChannel***
  Register base address

11.2 DMA Firmware driver API description

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

11.2.1 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_Handler() 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*

11.2.2 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()*

11.2.3 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()*

11.2.4 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()*

11.2.5 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 Channel.

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 Channel.
- **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 Channel.
- **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 (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 (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

11.3 DMA Firmware driver defines

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

11.3.1 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

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:

- Returns 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:
- Returns 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:
- Returns 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:
- Returns 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_TCx: Transfer complete flag
  – DMA_FLAG_HTx: Half transfer complete flag
  – DMA_FLAG_TEx: Transfer error flag
  – DMA_FLAG_GLx: Global interrupt flag Where x can be from 1 to 8 to select the DMA Channel x 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_TCx: Transfer complete flag
  – DMA_FLAG_HTx: Half transfer complete flag
  – DMA_FLAG_TEx: Transfer error flag
  – DMA_FLAG_GLx: Global interrupt flag Where x can be from 1 to 8 to select the DMA Channel x 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 not.

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_GL1

Channel 1 global flag

DMA_FLAG_TC1

Channel 1 transfer complete flag

DMA_FLAG_HT1

Channel 1 half transfer flag

DMA_FLAG_TE1

Channel 1 transfer error flag

DMA_FLAG_GL2

Channel 2 global flag

DMA_FLAG_TC2

Channel 2 transfer complete flag

DMA_FLAG_HT2

Channel 2 half transfer flag

DMA_FLAG_TE2

Channel 2 transfer error flag

DMA_FLAG_GL3

Channel 3 global flag

DMA_FLAG_TC3

Channel 3 transfer complete flag

DMA_FLAG_HT3

Channel 3 half transfer flag

DMA_FLAG_TE3
Channel 3 transfer error flag

DMA_FLAG_GL4
Channel 4 global flag

DMA_FLAG_TC4
Channel 4 transfer complete flag

DMA_FLAG_HT4
Channel 4 half transfer flag

DMA_FLAG_TE4
Channel 4 transfer error flag

DMA_FLAG_GL5
Channel 5 global flag

DMA_FLAG_TC5
Channel 5 transfer complete flag

DMA_FLAG_HT5
Channel 5 half transfer flag

DMA_FLAG_TE5
Channel 5 transfer error flag

DMA_FLAG_GL6
Channel 6 global flag

DMA_FLAG_TC6
Channel 6 transfer complete flag

DMA_FLAG_HT6
Channel 6 half transfer flag

DMA_FLAG_TE6
Channel 6 transfer error flag

DMA_FLAG_GL7
Channel 7 global flag

DMA_FLAG_TC7
Channel 7 transfer complete flag

DMA_FLAG_HT7
Channel 7 half transfer flag

DMA_FLAG_TE7
Channel 7 transfer error flag

DMA_FLAG_GL8
Channel 8 global flag

DMA_FLAG_TC8
Channel 8 transfer complete flag

DMA_FLAG_HT8

Channel 8 half transfer flag

DMA_FLAG_TE8

Channel 8 transfer error flag

TIM DMA Handle Index

TIM_DMA_ID_UPDATE

Index of the DMA handle used for Update DMA requests

TIM_DMA_ID_CC1

Index of the DMA handle used for Capture/Compare 1 DMA requests

TIM_DMA_ID_CC2

Index of the DMA handle used for Capture/Compare 2 DMA requests

TIM_DMA_ID_CC3

Index of the DMA handle used for Capture/Compare 3 DMA requests

TIM_DMA_ID_CC4

Index of the DMA handle used for Capture/Compare 4 DMA requests

DMA interrupt enable definitions

DMA_IT_TC

Transfer complete interrupt

DMA_IT_HT

Half Transfer interrupt

DMA_IT_TE

Transfer error interrupt

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_ADC1_DS

DMAMUX ADC Channel 0 request

DMA_REQUEST_SPI3_RX

DMAMUX SPI3 RX request

DMA_REQUEST_SPI3_TX

DMAMUX SPI3 TX request

DMA_REQUEST_I2C1_RX

DMAMUX I2C1 RX request

DMA_REQUEST_I2C1_TX

DMAMUX I2C1 TX request

DMA_REQUEST_USART1_RX

DMAMUX USART1 RX request

DMA_REQUEST_USART1_TX

DMAMUX USART1 TX request

DMA_REQUEST_LPUART1_RX

DMAMUX LP_UART1_RX request

DMA_REQUEST_LPUART1_TX

DMAMUX LP_UART1_RX request

DMAMUX_REQ_TIM2_CH1

DMAMUX TIM2 CH1 request

DMAMUX_REQ_TIM2_CH2

DMAMUX TIM2 CH3 request

DMAMUX_REQ_TIM2_CH3

DMAMUX TIM2 CH3 request

DMAMUX_REQ_TIM2_CH4

DMAMUX TIM2 CH4 request

DMAMUX_REQ_TIM2_UP

DMAMUX TIM2 UP request

DMAMUX_REQ_TIM16_CH1

DMAMUX TIM16 CH1 request

DMAMUX_REQ_TIM16_UP

DMAMUX TIM16 UP request

DMAMUX_REQ_TIM17_CH1

DMAMUX TIM17 CH1 request

DMAMUX_REQ_TIM17_UP

DMAMUX TIM17 UP request

DMA_MAX_REQUEST

12 HAL FLASH Generic Driver

12.1 FLASH Firmware driver registers structures

12.1.1 FLASH_EraseInitTypeDef

FLASH_EraseInitTypeDef is defined in the stm32wb0x_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 (max number of pages - 1)
- ***uint32_t FLASH_EraseInitTypeDef::NbPages***
  Number of pages to be erased. This parameter must be a value between 1 and (max number of pages - value of initial page)

12.1.2 FLASH_ProcessTypeDef

FLASH_ProcessTypeDef is defined in the stm32wb0x_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***
  FLASH locking object
- ***uint32_t FLASH_ProcessTypeDef::ErrorCode***
  FLASH error code
- ***uint32_t FLASH_ProcessTypeDef::ProcedureOnGoing***
  Internal variable to indicate which procedure is ongoing or not in IT context
- ***uint32_t FLASH_ProcessTypeDef::Address***
  Internal variable to save address selected for program in IT context
- ***uint32_t FLASH_ProcessTypeDef::Page***
  Internal variable to define the current page which is erasing in IT context
- ***uint32_t FLASH_ProcessTypeDef::NbPagesToErase***
  Internal variable to save the remaining pages to erase in IT context

12.1.3 FLASH_PageProtectionTypeDef

FLASH_PageProtectionTypeDef is defined in the stm32wb0x_hal_flash.h

Data Fields

- ***uint32_t pageProt_0***
- ***uint32_t pageProt_1***
- ***FunctionalState state***

Field Documentation

- ***uint32_t FLASH_PageProtectionTypeDef::pageProt_0***
  For cut 1.0 indicate the group to configure.
  – This parameter can be a value of ***FLASH_PAGEPROT_GROUP*** For cut 2.0 it is a bitfield built as follows:
  – bit[6:0] = SIZE: number of 2kB pages to protect including the starting page **pageProt_0** param (provided in the offset bits)
  – bit[7] = reserved
  – bit[14:8] = OFFSET: page number to start the write protection (value between 0 and 0x7F)
  – bit[15] = reserved
  – bit[22:16] = SIZE: same meaning of bit[6:0]
  – bit[23] = reserved
  – bit[30:24] = OFFSET: same meaning of bit[22:16]
  – bit[31] = reserved
- ***uint32_t FLASH_PageProtectionTypeDef::pageProt_1***
  Valid only for cut 2.0 same meaning of **pageProt_0** parameter
- ***FunctionalState FLASH_PageProtectionTypeDef::state***
  The new state of the group. This parameter can be a value of FunctionalState enumeration

12.2 FLASH Firmware driver API description

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

12.2.1 FLASH peripheral features

The flash memory interface implements the erase and program flash memory operations and write/erase protection mechanisms.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections

12.2.2 How to use this driver

This driver provides functions and macros to configure and program the FLASH memory.

1. Flash Memory IO Programming functions:
   – Program functions: word and burst program
   – There are two modes of programming:
     ◦ Polling mode using HAL_FLASH_Program() function
     ◦ Interrupt mode using HAL_FLASH_Program_IT() function
     ◦ Polling mode using HAL_FLASH_BurstProgram() function
     ◦ Interrupt mode using HAL_FLASH_BurstProgram_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()

In addition to these functions, this driver includes a set of macros allowing to handle the following operations:

- Set the wait state latency
- Enable/Disable the flash interrupts
- Monitor the flash flags status

12.2.3 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()*

12.2.4 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()*

12.2.5 Detailed description of functions

HAL_FLASH_Program

Function name

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

Function description

Program a data 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 word program and the address where are stored the data for the burst 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, uint32_t Data)

Function description

Program a word or burst 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_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_ILLCMD FLASH Illegal command

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

12.3 FLASH Firmware driver defines

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

12.3.1 FLASH

FLASH

PROTECTION_level FLASH Protection level

PROT_LEVEL_NONE

PROT_LEVEL_1

PROT_LEVEL_2

PROT_LEVEL_3

FLASH Command

FLASH_CMD_ERASE_PAGES

Flash pages erase command

FLASH_CMD_MASSERASE

Flash mass erase command

FLASH_CMD_WRITE

Flash write command

FLASH_CMD_MASSREAD

Flash mass read command

FLASH_CMD_SLEEP

Flash in sleep mode command

FLASH_CMD_WAKEUP

Flash wakeup command

FLASH_CMD_BURSTWRITE

Flash burst write command

FLASH_CMD_OTPWRITE

Flash OTP write command

FLASH_CMD_KEYWRITE

Flash keywrite command

FLASH Configuration Macros

__HAL_FLASH_SET_WAIT_STATES

Description:

- Set the FLASH Wait States.

Parameters:

- __WAIT_STATES__: FLASH WAIT States This parameter can be one of the following values :
  - FLASH_WAIT_STATES_0 FLASH Zero wait state
  - FLASH_WAIT_STATES_1 FLASH One wait state

Return value:

- None

__HAL_FLASH_GET_WAIT_STATES

Description:

- Get the FLASH Wait States.

Return value:

- FLASH: Wait States Returned value can be one of the following values :
  - FLASH_WAIT_STATES_0 FLASH Zero wait state
  - FLASH_WAIT_STATES_1 FLASH One wait state

__HAL_FLASH_BURSTWRITE_ENABLE

Description:

- Enable the FLASH burst write operations.

Return value:

- None

__HAL_FLASH_BURSTWRITE_DISABLE

Description:

- Disable the FLASH burst write operations.

Return value:

- None

__HAL_FLASH_SET_VECT_TABLE

Description:

- Set the FLASH Interrupt Vector Table location.

Parameters:

- __VECT_TABLE__: Vector Table Configuration This parameter can be one of the following values :
  - FLASH_CONFIG_IRQ_VECT_TABLE_FLASH Vector Table located in FLASH
  - FLASH_CONFIG_IRQ_VECT_TABLE_RAM Vector Table located in RAM

Return value:

- None

__HAL_FLASH_GET_VECT_TABLE

Description:

- Get the FLASH Interrupt Vector Table location.

Return value:

- Vector: Table Configuration Returned value can be one of the following values :
  - FLASH_CONFIG_IRQ_VECT_TABLE_FLASH Vector Table located in FLASH
  - FLASH_CONFIG_IRQ_VECT_TABLE_RAM Vector Table located in RAM

FLASH Configuration Interrupt Vector Table

FLASH_CONFIG_IRQ_VECT_TABLE_FLASH

Vector Table located in FLASH

FLASH_CONFIG_IRQ_VECT_TABLE_RAM

Vector Table located in RAM

FLASH Error

HAL_FLASH_ERROR_NONE

HAL_FLASH_ERROR_OP

HAL_FLASH_ERROR_ILLCMD

FLASH Flags Macros

__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_CMDDONE FLASH command done flag
  - FLASH_FLAG_CMDSTART FLASH command started flag
  - FLASH_FLAG_CMDERR FLASH command error flag
  - FLASH_FLAG_ILLCMD FLASH illegal command flag
  - FLASH_FLAG_READOK FLASH mass read flag

Return value:

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

__HAL_FLASH_CLEAR_FLAG

Description:

- Clear the FLASH's pending flags.

Parameters:

- __FLAG__: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  - FLASH_FLAG_CMDDONE FLASH command done flag
  - FLASH_FLAG_CMDSTART FLASH command started flag
  - FLASH_FLAG_CMDERR FLASH command error flag
  - FLASH_FLAG_ILLCMD FLASH illegal command flag
  - FLASH_FLAG_READOK FLASH mass read flag

Return value:

- None

FLASH Flags Definition

FLASH_FLAG_CMDDONE

FLASH command done flag

FLASH_FLAG_CMDSTART

FLASH command started flag

FLASH_FLAG_CMDERR

FLASH command error flag

FLASH_FLAG_ILLCMD

FLASH illegal command flag

FLASH_FLAG_READOK

FLASH mass read command flag

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_CMDDONE Command Done Interrupt
  – FLASH_IT_CMDSTART Command Started Interrupt
  – FLASH_IT_CMDERR Command Error Interrupt
  – FLASH_IT_ILLCMD Illegal Command Interrupt
  – FLASH_IT_READOK Mass read Command 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_CMDDONE Command Done Interrupt
  – FLASH_IT_CMDSTART Command Started Interrupt
  – FLASH_IT_CMDERR Command Error Interrupt
  – FLASH_IT_ILLCMD Illegal Command Interrupt
  – FLASH_IT_READOK Mass read Command Interrupt

Return value:

- none

__HAL_FLASH_GET_IT

Description:

- Check whether the specified FLASH interrupt has occurred or not.

Parameters:

- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  – FLASH_IT_CMDDONE Command Done Interrupt
  – FLASH_IT_CMDSTART Command Started Interrupt
  – FLASH_IT_CMDERR Command Error Interrupt
  – FLASH_IT_ILLCMD Illegal Command Interrupt
  – FLASH_IT_READOK Mass read Command Interrupt

Return value:

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

__HAL_FLASH_CLEAR_IT

Description:
- Clear the specified FLASH interrupt flag.

Parameters:
- __INTERRUPT__: FLASH interrupt This parameter can be any combination of the following values:
  – FLASH_IT_CMDDONE Command Done Interrupt
  – FLASH_IT_CMDSTART Command Started Interrupt
  – FLASH_IT_CMDERR Command Error Interrupt
  – FLASH_IT_ILLCMD Illegal Command Interrupt
  – FLASH_IT_READOK Mass read Command Interrupt

Return value:
- None

FLASH Interrupts Definition

FLASH_IT_CMDDONE

FLASH command done Interrupt source

FLASH_IT_CMDSTART

FLASH command started Interrupt source

FLASH_IT_CMDERR

FLASH command error Interrupt source

FLASH_IT_ILLCMD

FLASH illegal command Interrupt source

FLASH_IT_READOK

FLASH mass read command Interrupt source

FLASH Keys

FLASH_KEY0

Flash key0

FLASH_KEY1

Flash key1

FLASH_KEY2

Flash key2

FLASH_DATAKEY2

FLASH_DATAKEY3

FLASH write/page erase protection for group

FLASH_PAGEPROT_GROUP_15

write/page erase protection for group 15 containing flash pages 120 to 127

FLASH_PAGEPROT_GROUP_14

write/page erase protection for group 14 containing flash pages 112 to 119

FLASH_PAGEPROT_GROUP_13

write/page erase protection for group 13 containing flash pages 104 to 111

FLASH_PAGEPROT_GROUP_12

write/page erase protection for group 12 containing flash pages 96 to 103

FLASH_PAGEPROT_GROUP_11

write/page erase protection for group 11 containing flash pages 88 to 95

FLASH_PAGEPROT_GROUP_10

write/page erase protection for group 10 containing flash pages 80 to 87

FLASH_PAGEPROT_GROUP_9

write/page erase protection for group 9 containing flash pages 72 to 79

FLASH_PAGEPROT_GROUP_8

write/page erase protection for group 8 containing flash pages 64 to 71

FLASH_PAGEPROT_GROUP_7

write/page erase protection for group 7 containing flash pages 56 to 63

FLASH_PAGEPROT_GROUP_6

write/page erase protection for group 6 containing flash pages 48 to 55

FLASH_PAGEPROT_GROUP_5

write/page erase protection for group 5 containing flash pages 40 to 47

FLASH_PAGEPROT_GROUP_4

write/page erase protection for group 4 containing flash pages 32 to 39

FLASH_PAGEPROT_GROUP_3

write/page erase protection for group 3 containing flash pages 24 to 31

FLASH_PAGEPROT_GROUP_2

write/page erase protection for group 2 containing flash pages 16 to 23

FLASH_PAGEPROT_GROUP_1

write/page erase protection for group 1 containing flash pages 8 to 15

FLASH_PAGEPROT_GROUP_0

write/page erase protection for group 0 containing flash pages 0 to 7

FLASH Erase Type

FLASH_TYPEERASE_PAGES

Pages erase

FLASH_TYPEERASE_MASSERASE

Flash mass erase

FLASH Program Type

FLASH_TYPEPROGRAM_WORD

FLASH_TYPEPROGRAM_BURST

FLASH Wait States

FLASH_WAIT_STATES_0

FLASH Zero wait state

FLASH_WAIT_STATES_1

FLASH One wait state

13 HAL FLASH Extension Driver

13.1 FLASHEx Firmware driver API description

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

13.1.1 Flash Extended features

The FLASH interface contains the following additional features

- Single bank memory organization
- Readout protection
- WRERP protection

13.1.2 How to use this driver

This driver provides functions to configure and program the FLASH memory. It includes

1. Flash Memory Erase 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. OTP Memory Programming functions:
   – There are two modes of OTP program :
      ◦ Polling Mode using HAL_FLASHEx_OTPWRite()
      ◦ Interrupt Mode using HAL_FLASHEx_OTPWrite_IT()

13.1.3 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_OTPWrite()*
- *HAL_FLASHEx_OTPWrite_IT()*
- *HAL_FLASHEx_PageProtection()*
- *HAL_FLASHEx_Protection_Config()*
- *HAL_FLASHEx_GetProtection_Level()*
- *HAL_FLASHEx_MassRead()*
- *HAL_FLASHEx_LFSRSignature()*

13.1.4 Detailed description of functions

HAL_FLASHEx_Erase

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)

Function description

Perform a mass erase or erase the specified FLASH memory pages.

Parameters

- **pEraseInit:** Pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
- **PageError:** Pointer to variable that contains the configuration information on faulty page in case of error (0xFFFFFFFF means that all the pages have been correctly erased)

Return values

- **HAL:** Status

HAL_FLASHEx_Erase_IT

Function name

HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)

Function description

Perform a mass erase or erase the specified FLASH memory pages with interrupt enabled.

Parameters

- **pEraseInit:** Pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.

Return values

- **HAL:** Status

HAL_FLASHEx_OTPWrite

Function name

HAL_StatusTypeDef HAL_FLASHEx_OTPWrite (uint32_t Address, uint32_t Data)

Function description

Program a word at a specified OTP address.

Parameters

- **Address:** Specifies the address to be programmed.
- **Data:** Specifies the data to be programmed.

Return values

- **HAL_StatusTypeDef:** HAL Status

HAL_FLASHEx_OTPWrite_IT

Function name

HAL_StatusTypeDef HAL_FLASHEx_OTPWrite_IT (uint32_t Address, uint32_t Data)

Function description

Program a word at a specified OTP address with interrupt enabled.

Parameters

- **Address:** Specifies the address to be programmed.
- **Data:** Specifies the data to be programmed.

Return values

- **HAL:** Status

HAL_FLASHEx_PageProtection

Function name

HAL_StatusTypeDef HAL_FLASHEx_PageProtection (FLASH_PageProtectionTypeDef * pageProtectionStruct)

Function description

Page Protection function.

Parameters

- **pageProtectionStruct:** Pointer to a FLASH_PageProtectionTypeDef enumeration. Bitfield with the write/ page erase protection for group of 8 pages (group 0: Flash pages 0 to 7, group 1: Flash pages 8 to 15... i.e. if pageProt=0x0C the function will protect pages from 16 to 31).

Return values

- **HAL:** Status

HAL_FLASHEx_GetProtection_Level

Function name

uint32_t HAL_FLASHEx_GetProtection_Level (void)

Function description

Return the FLASH Protection level.

Return values

- **FLASH:** Protection Level: This return value can be one of the following values:
  – PROT_LEVEL_NONE
  – PROT_LEVEL_1
  – PROT_LEVEL_2
  – PROT_LEVEL_3

HAL_FLASHEx_Protection_Config

Function name

HAL_StatusTypeDef HAL_FLASHEx_Protection_Config (uint32_t ProtectionLevel)

Function description

Set Protection level configuration.

Parameters

- **ProtectionLevel:** Level of protection This parameter can be a combination of all the following values:
  – PROT_LEVEL_NONE
  – PROT_LEVEL_1
  – PROT_LEVEL_2
  – PROT_LEVEL_3

Return values

- **None:**

HAL_FLASHEx_MassRead

Function name

HAL_StatusTypeDef HAL_FLASHEx_MassRead (uint32_t pattern)

Function description

Perform a mass read.

Parameters

- **pattern:** Specifies the expected value.

Return values

- **HAL_StatusTypeDef:** HAL Status

HAL_FLASHEx_LFSRSignature

Function name

HAL_StatusTypeDef HAL_FLASHEx_LFSRSignature (uint32_t * LFSR_Result)

Function description

Perform a LFSR signature issued by a MASSREAD command.

Return values

- **HAL:** Status

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 memory - 1)

Return values

- **None:**

14 HAL GPIO Generic Driver

14.1 GPIO Firmware driver registers structures

14.1.1 GPIO_InitTypeDef

GPIO_InitTypeDef is defined in the stm32wb0x_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 ***Section 15.1.1: GPIOEx_Alternate_function_selection***

14.2 GPIO Firmware driver API description

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

14.2.1 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
  – Output mode
  – Alternate function mode
  – External interrupt/event lines
- Open-drain and analog features are not available on all IO ports, refer to reference manual for available pins.
- During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input pull-up 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 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.

14.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Modify the control of the product's pull-up/-down of the IO ports from PWR registers (which is the default state) using either LL_PWR_EnableGPIOPullUp or LL_PWR_EnableGPIOPullDown APIs depending on the requested pull mode. This will allow to switch the control to the GPIO register.
3. 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).
4. 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().
5. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
6. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
7. To set the level of several pins and reset level of several other pins in same cycle, use HAL_GPIO_WriteMultipleStatePin().
8. To lock pin configuration until next reset use HAL_GPIO_LockPin().
9. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input pull-up mode (except JTAG pins).
10. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PB12 and PB13, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
11. The HSE oscillator pins OSC_IN/OSC_OUT are dedicated oscillator IOs and can not be used as general purpose.

14.2.3 Initialization and de-initialization functions

This section contains the following APIs:

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

14.2.4 Detailed description of functions

HAL_GPIO_Init

Function name

void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)

Function description

Initialize the GPIOx peripheral according to the specified parameters in the GPIO_Init.

Parameters

- **GPIOx:** where x can be (A..B) to select the GPIO peripheral for STM32WB0x family
- **GPIO_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..B) to select the GPIO peripheral for STM32WB0x family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

Return values

- **None:**

HAL_GPIO_ReadPin

Function name

GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Read the specified input port pin.

Parameters

- **GPIOx:** where x can be (A..B) to select the GPIO peripheral for STM32WB0x family
- **GPIO_Pin:** specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).

Return values

- **The:** input port pin value.

HAL_GPIO_WritePin

Function name

void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

Function description

Set or clear the selected data port bit.

Parameters

- **GPIOx:** where x can be (A..B) to select the GPIO peripheral for STM32WB0x family
- **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
- **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 STM32WLxx 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..B) to select the GPIO peripheral for STM32WB0x 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..B) to select the GPIO peripheral for STM32WB0x 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 (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

Handle EXTI interrupt request.

Parameters

- **GPIOx:** where x can be (A..B) to select the GPIO peripheral
- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

HAL_GPIO_EXTI_Callback

Function name

void HAL_GPIO_EXTI_Callback (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

Function description

EXTI line detection callback.

Parameters

- **GPIOx:** where x can be (A..B) to select the GPIO peripheral
- **GPIO_Pin:** Specifies the port pin connected to corresponding EXTI line.

Return values

- **None:**

14.3 GPIO Firmware driver defines

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

14.3.1 GPIO

GPIO

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Check whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_PORT__: specifies the EXTI port to check. This parameter can be GPIOA or GPIOB.
- __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's line pending flags.

Parameters:

- __EXTI_PORT__: specifies the EXTI port to check. This parameter can be GPIOA or GPIOB.
- __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

__HAL_GPIO_EXTI_GET_IT

Description:

- Check whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_PORT__: specifies the EXTI port to check. This parameter can be GPIOA or GPIOB.
- __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_PORT__: specifies the EXTI port to check. This parameter can be GPIOA or GPIOB.
- __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

GPIO mode

GPIO_MODE_INPUT

Input Floating Mode

GPIO_MODE_OUTPUT_PP

Output Push Pull Mode

GPIO_MODE_OUTPUT_OD

Output Open Drain Mode

GPIO_MODE_AF_PP

Alternate Function Push Pull Mode

GPIO_MODE_AF_OD

Alternate Function Open Drain Mode

GPIO_MODE_ANALOG

Analog Mode

GPIO_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge trigger detection

GPIO_MODE_IT_LEVEL_HIGH

External Interrupt Mode with High level trigger detection

GPIO_MODE_IT_LEVEL_LOW

External Interrupt Mode with Low level trigger detection

GPIO_MODE_EVT_RISING

External Event Mode with Rising edge trigger detection

GPIO_MODE_EVT_FALLING

External Event Mode with Falling edge trigger detection

GPIO_MODE_EVT_RISING_FALLING

External Event Mode with Rising/Falling edge trigger detection

GPIO_MODE_EVT_LEVEL_HIGH

External Event Mode with High level trigger detection

GPIO_MODE_EVT_LEVEL_LOW

External Event Mode with Low level trigger detection

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

15 HAL GPIO Extension Driver

15.1 GPIOEx Firmware driver defines

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

15.1.1 GPIOEx

GPIOEx

GPIOEx Get Port Index

GPIO_GET_INDEX

16 HAL I2C Generic Driver

16.1 I2C Firmware driver registers structures

16.1.1 I2C_InitTypeDef

I2C_InitTypeDef is defined in the stm32wb0x_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***

16.1.2 __I2C_HandleTypeDef

__I2C_HandleTypeDef is defined in the stm32wb0x_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 sequential 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

16.2 I2C Firmware driver API description

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

16.2.1 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

16.2.2 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()*

16.2.3 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()*

16.2.4 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()*

16.2.5 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

16.3 I2C Firmware driver defines

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

16.3.1 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

17 HAL I2C Extension Driver

17.1 I2CEx Firmware driver API description

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

17.1.1 I2C peripheral Extended features

Comparing to other previous devices, the I2C interface for STM32WB0x devices contains the following additional features

- Possibility to disable or enable Analog Noise Filter
- Use of a configured Digital Noise Filter
- Disable or enable Fast Mode Plus

17.1.2 How to use this driver

This driver provides functions to:

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 fast mode plus driving capability using the functions :
   – HAL_I2CEx_EnableFastModePlus()
   – HAL_I2CEx_DisableFastModePlus()

17.1.3 Filter Mode Functions

This section provides functions allowing to:

- Configure Noise Filters

This section contains the following APIs:

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

17.1.4 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()*

17.1.5 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_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:**

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:**

17.2 I2CEx Firmware driver defines

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

17.2.1 I2CEx

I2CEx

I2C Extended Analog Filter

I2C_ANALOGFILTER_ENABLE

I2C_ANALOGFILTER_DISABLE

I2C Extended Fast Mode Plus

I2C_FASTMODEPLUS_PA0

Enable Fast Mode Plus on PA0

I2C_FASTMODEPLUS_PA1

Enable Fast Mode Plus on PA1

I2C_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

I2C_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

18 HAL I2S Generic Driver

18.1 I2S Firmware driver registers structures

18.1.1 I2S_InitTypeDef

I2S_InitTypeDef is defined in the stm32wb0x_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***

18.1.2 I2S_HandleTypeDef

I2S_HandleTypeDef is defined in the stm32wb0x_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***

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***

18.2 I2S Firmware driver API description

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

18.2.1 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:

- *External clock source is configured after setting correctly the define constant EXTERNAL_CLOCK_VALUE in the stm32wb0x_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.

18.2.2 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()*

18.2.3 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()*

18.2.4 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()*

18.2.5 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_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

18.3 I2S Firmware driver defines

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

18.3.1 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_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

19 HAL IRDA Generic Driver

19.1 IRDA Firmware driver registers structures

19.1.1 IRDA_InitTypeDef

IRDA_InitTypeDef is defined in the stm32wb0x_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***.

19.1.2 __IRDA_HandleTypeDef

__IRDA_HandleTypeDef is defined in the stm32wb0x_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

19.2 IRDA Firmware driver API description

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

19.2.1 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

19.2.2 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.

19.2.3 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()*

19.2.4 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()*

19.2.5 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()*

19.2.6 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

19.3 IRDA Firmware driver defines

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

19.3.1 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

20 HAL IRDA Extension Driver

20.1 IRDAEx Firmware driver defines

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

20.1.1 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

21 HAL IWDG Generic Driver

21.1 IWDG Firmware driver registers structures

21.1.1 IWDG_InitTypeDef

IWDG_InitTypeDef is defined in the stm32wb0x_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

21.1.2 IWDG_HandleTypeDef

IWDG_HandleTypeDef is defined in the stm32wb0x_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

21.2 IWDG Firmware driver API description

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

21.2.1 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 at 32 KHz (LSI): ~125 µs / ~32.7 s The IWDG timeout may vary due to LSI clock frequency dispersion. STM32WB0x 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).

21.2.2 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

21.2.3 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()*

21.2.4 IO operation functions

This section provides functions allowing to:
- Refresh the IWDG.

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

21.2.5 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

21.3 IWDG Firmware driver defines

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

21.3.1 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

22 HAL PKA Generic Driver

22.1 PKA Firmware driver registers structures

22.1.1 __PKA_HandleTypeDef

__PKA_HandleTypeDef is defined in the stm32wb0x_hal_pka.h

Data Fields

- ***PKA_TypeDef * Instance***
- ***__IO HAL_PKA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***void(* OperationCpltCallback***
- ***void(* ErrorCallback***
- ***void(* MspInitCallback***
- ***void(* MspDeInitCallback***

Field Documentation

- ***PKA_TypeDef* __PKA_HandleTypeDef::Instance***
  Register base address
- ***__IO HAL_PKA_StateTypeDef __PKA_HandleTypeDef::State***
  PKA state
- ***__IO uint32_t __PKA_HandleTypeDef::ErrorCode***
  PKA Error code
- ***void(* __PKA_HandleTypeDef::OperationCpltCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA End of operation callback
- ***void(* __PKA_HandleTypeDef::ErrorCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Error callback
- ***void(* __PKA_HandleTypeDef::MspInitCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Msp Init callback
- ***void(* __PKA_HandleTypeDef::MspDeInitCallback)(struct __PKA_HandleTypeDef *hpka)***
  PKA Msp DeInit callback

22.1.2 PKA_ECCMulInTypeDef

PKA_ECCMulInTypeDef is defined in the stm32wb0x_hal_pka.h

Data Fields

- ***uint32_t scalarMulSize***
- ***uint32_t modulusSize***
- ***const uint8_t * pointX***
- ***const uint8_t * pointY***
- ***const uint8_t * scalarMul***

Field Documentation

- ***uint32_t PKA_ECCMulInTypeDef::scalarMulSize***
  Number of element in scalarMul array
- ***uint32_t PKA_ECCMulInTypeDef::modulusSize***
  Number of element in modulus, coefA, pointX and pointY arrays
- ***const uint8_t* PKA_ECCMulInTypeDef::pointX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::pointY***
  Pointer to point P coordinate yP (Array of modulusSize elements)
- ***const uint8_t* PKA_ECCMulInTypeDef::scalarMul***
  Pointer to scalar multiplier k (Array of scalarMulSize elements)

22.1.3 PKA_ECCMulOutTypeDef

PKA_ECCMulOutTypeDef is defined in the stm32wb0x_hal_pka.h

Data Fields

- ***uint8_t * ptX***
- ***uint8_t * ptY***

Field Documentation

- ***uint8_t* PKA_ECCMulOutTypeDef::ptX***
  Pointer to point P coordinate xP (Array of modulusSize elements)
- ***uint8_t* PKA_ECCMulOutTypeDef::ptY***
  Pointer to point P coordinate yP (Array of modulusSize elements)

22.2 PKA Firmware driver API description

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

22.2.1 How to use this driver

The PKA HAL driver can be used as follows:

1. Declare a PKA_HandleTypeDef handle structure, for example: PKA_HandleTypeDef hpka;
2. Initialize the PKA low level resources by implementing the HAL_PKA_MspInit() API:
   a. Enable the PKA interface clock
   b. NVIC configuration if you need to use interrupt process
      - Configure the PKA interrupt priority
      - Enable the NVIC PKA IRQ Channel
3. Initialize the PKA registers by calling the HAL_PKA_Init() API which trig HAL_PKA_MspInit().
4. Fill entirely the input structure corresponding to your operation: For instance: PKA_ECCMulInTypeDef for HAL_PKA_ECCMul().
5. Execute the operation (in polling or interrupt) and check the returned value.
6. Retrieve the result of the operation (For instance, HAL_PKA_ModExp_GetResult for HAL_PKA_ModExp operation). The function to gather the result is different for each kind of operation. The correspondence can be found in the following section.
7. Call the function HAL_PKA_DeInit() to restore the default configuration which trig HAL_PKA_MspDeInit().

High level operation

- Input structure requires buffers as uint8_t array.
- Output structure requires buffers as uint8_t array.
- ECC Scalar Multiplication operation for stm32wb05, stm32wb06, stm32wb07, stm32wb09:
- ECC Scalar Multiplication using:
  – HAL_PKA_ECCMul().
  – HAL_PKA_ECCMul_IT().
  – HAL_PKA_ECCMul_GetResult() to retrieve the result of the operation.
- Others operation for stm32wb05, stm32wb09 only:
- Modular exponentiation using:
  – HAL_PKA_ModExp().
  – HAL_PKA_ModExp_IT().
  – HAL_PKA_ModExpFastMode().
  – HAL_PKA_ModExpFastMode_IT().
  – HAL_PKA_ModExp_GetResult() to retrieve the result of the operation.
- RSA Chinese Remainder Theorem (CRT) using:
  – HAL_PKA_RSACRTExp().
  – HAL_PKA_RSACRTExp_IT().
  – HAL_PKA_RSACRTExp_GetResult() to retrieve the result of the operation.

- ECC Point Check using:
  - HAL_PKA_PointCheck().
  - HAL_PKA_PointCheck_IT().
  - HAL_PKA_PointCheck_IsOnCurve() to retrieve the result of the operation.
- ECDSA Sign
  - HAL_PKA_ECDSASign().
  - HAL_PKA_ECDSASign_IT().
  - HAL_PKA_ECDSASign_GetResult() to retrieve the result of the operation.
- ECDSA Verify
  - HAL_PKA_ECDSAVerif().
  - HAL_PKA_ECDSAVerif_IT().
  - HAL_PKA_ECDSAVerif_IsValidSignature() to retrieve the result of the operation.
- ECC Scalar Multiplication using:
  - HAL_PKA_ECCMulFastMode().
  - HAL_PKA_ECCMulFastMode_IT().
- High level operation for stm32wb09 only:
- ECC double base ladder using:
  - HAL_PKA_ECCDoubleBaseLadder().
  - HAL_PKA_ECCDoubleBaseLadder_IT().
  - HAL_PKA_ECCDoubleBaseLadder_GetResult() to retrieve the result of the operation.
- ECC complete addition using:
  - HAL_PKA_ECCCompleteAddition().
  - HAL_PKA_ECCCompleteAddition_IT().
  - HAL_PKA_ECCCompleteAddition_GetResult() to retrieve the result of the operation.

Low level operation for stm32wb05, stm32wb09 only

- Input structure requires buffers as uint32_t array.
- Output structure requires buffers as uint32_t array.
- Arithmetic addition using:
  - HAL_PKA_Add().
  - HAL_PKA_Add_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation. The resulting size can be the input parameter or the input parameter size + 1 (overflow).
- Arithmetic subtraction using:
  - HAL_PKA_Sub().
  - HAL_PKA_Sub_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Arithmetic multiplication using:
  - HAL_PKA_Mul().
  - HAL_PKA_Mul_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Comparison using:
  - HAL_PKA_Cmp().
  - HAL_PKA_Cmp_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular addition using:
  - HAL_PKA_ModAdd().
  - HAL_PKA_ModAdd_IT().
  - HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.

- Modular subtraction using:
  – HAL_PKA_ModSub().
  – HAL_PKA_ModSub_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular inversion using:
  – HAL_PKA_ModInv().
  – HAL_PKA_ModInv_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Modular reduction using:
  – HAL_PKA_ModRed().
  – HAL_PKA_ModRed_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.
- Montgomery multiplication using:
  – HAL_PKA_MontgomeryMul().
  – HAL_PKA_MontgomeryMul_IT().
  – HAL_PKA_Arithmetic_GetResult() to retrieve the result of the operation.

Montgomery parameter for stm32wb05, stm32wb09 only

Polling mode operation

- When an operation is started in polling mode, the function returns when:
  – A timeout is encounter.
  – The operation is completed.

Interrupt mode operation

- Add HAL_PKA_IRQHandler to the IRQHandler of PKA.
- Enable the IRQ using HAL_NVIC_EnableIRQ().
- When an operation is started in interrupt mode, the function returns immediately.
- When the operation is completed, the callback HAL_PKA_OperationCpltCallback is called.
- When an error is encountered, the callback HAL_PKA_ErrorCallback is called.
- To stop any operation in interrupt mode, use HAL_PKA_Abort().

Utilities

- To clear the PKA RAM, use HAL_PKA_RAMReset().
- To get current state, use HAL_PKA_GetState().
- To get current error, use HAL_PKA_GetError().

Callback registration

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

Function HAL_PKA_RegisterCallback() allows to register following callbacks:

- OperationCpltCallback : callback for End of operation.
- ErrorCallback : callback for error detection.
- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit. This function takes as parameters the HAL peripheral handle, the Callback ID and a pointer to the user callback function.

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

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

- OperationCpltCallback : callback for End of operation.
- ErrorCallback : callback for error detection.

- MspInitCallback : callback for Msp Init.
- MspDeInitCallback : callback for Msp DeInit.

By default, after the HAL_PKA_Init() and when the state is HAL_PKA_STATE_RESET all callbacks are set to the corresponding weak functions: examples HAL_PKA_OperationCpltCallback(), HAL_PKA_ErrorCallback(). Exception done for MspInit and MspDeInit functions that are reset to the legacy weak functions in the HAL_PKA_Init()/ HAL_PKA_DeInit() only when these callbacks are null (not registered beforehand).

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

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

Then, the user first registers the MspInit/MspDeInit user callbacks using HAL_PKA_RegisterCallback() before calling HAL_PKA_DeInit() or HAL_PKA_Init() function.

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

22.2.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and deinitialize the PKAx peripheral:

- User must implement HAL_PKA_MspInit() function in which he configures all related peripherals resources (CLOCK, IT and NVIC).
- Call the function HAL_PKA_Init() to configure the device.
- Call the function HAL_PKA_DeInit() to restore the default configuration of the selected PKAx peripheral.

This section contains the following APIs:

- *HAL_PKA_Init()*
- *HAL_PKA_DeInit()*
- *HAL_PKA_MspInit()*
- *HAL_PKA_MspDeInit()*
- *HAL_PKA_RegisterCallback()*
- *HAL_PKA_UnRegisterCallback()*

22.2.3 IO operation functions

This subsection provides a set of functions allowing to manage the PKA operations.

1. There are two modes of operation:
   - Blocking mode : The operation is performed in the polling mode. These functions return when data operation is completed.
   - No-Blocking mode : The operation is performed using Interrupts. These functions return immediately. The end of the operation is indicated by HAL_PKA_ErrorCallback in case of error. The end of the operation is indicated by HAL_PKA_OperationCpltCallback in case of success. To stop any operation in interrupt mode, use HAL_PKA_Abort().

2. Blocking mode functions are : (+) Operation for stm32wb05, stm32wb06, stm32wb07, stm32wb09:
   – HAL_PKA_ECCMul()
   – HAL_PKA_ECCMul_GetResult(); (+) Operation for stm32wb05, stm32wb09 only:
   – HAL_PKA_ModExp()
   – HAL_PKA_ModExpFastMode()
   – HAL_PKA_ModExp_GetResult();
   – HAL_PKA_ECDSASign()
   – HAL_PKA_ECDSASign_GetResult();
   – HAL_PKA_ECDSAVerif()
   – HAL_PKA_ECDSAVerif_IsValidSignature();
   – HAL_PKA_RSACRTExp()
   – HAL_PKA_RSACRTExp_GetResult();
   – HAL_PKA_PointCheck()
   – HAL_PKA_PointCheck_IsOnCurve();
   – HAL_PKA_ECCMulFastMode()
   – HAL_PKA_Add()
   – HAL_PKA_Sub()
   – HAL_PKA_Cmp()
   – HAL_PKA_Mul()
   – HAL_PKA_ModAdd()
   – HAL_PKA_ModSub()
   – HAL_PKA_ModInv()
   – HAL_PKA_ModRed()
   – HAL_PKA_MontgomeryMul()
   – HAL_PKA_Arithmetic_GetResult(P);
   – HAL_PKA_MontgomeryParam()
   – HAL_PKA_MontgomeryParam_GetResult(); (+) Operation for stm32wb09 only:
   – HAL_PKA_ECCDoubleBaseLadder()
   – HAL_PKA_ECCDoubleBaseLadder_GetResult();
   – HAL_PKA_ECCCompleteAddition()
   – HAL_PKA_ECCCompleteAddition_GetResult();

3. No-Blocking mode functions with Interrupt are : (+) Operation for stm32wb05, stm32wb06, stm32wb07, stm32wb09:
   – HAL_PKA_ECCMul_IT();
   – HAL_PKA_ECCMul_GetResult();
   – HAL_PKA_Abort(); (+) Operation for stm32wb05, stm32wb09:
   – HAL_PKA_ModExp_IT();
   – HAL_PKA_ModExpFastMode_IT();
   – HAL_PKA_ModExp_GetResult();
   – HAL_PKA_ECDSASign_IT();
   – HAL_PKA_ECDSASign_GetResult();
   – HAL_PKA_ECDSAVerif_IT();
   – HAL_PKA_ECDSAVerif_IsValidSignature();
   – HAL_PKA_RSACRTExp_IT();
   – HAL_PKA_RSACRTExp_GetResult();
   – HAL_PKA_PointCheck_IT();
   – HAL_PKA_PointCheck_IsOnCurve();
   – HAL_PKA_ECCMulFastMode_IT();
   – HAL_PKA_Add_IT();
   – HAL_PKA_Sub_IT();
   – HAL_PKA_Cmp_IT();
   – HAL_PKA_Mul_IT();
   – HAL_PKA_ModAdd_IT();
   – HAL_PKA_ModSub_IT();
   – HAL_PKA_ModInv_IT();
   – HAL_PKA_ModRed_IT();
   – HAL_PKA_MontgomeryMul_IT();
   – HAL_PKA_Arithmetic_GetResult();
   – HAL_PKA_MontgomeryParam_IT();
   – HAL_PKA_MontgomeryParam_GetResult(); (+) Operation for stm32wb09:
   – HAL_PKA_ECCDoubleBaseLadder_IT()
   – HAL_PKA_ECCDoubleBaseLadder_GetResult();
   – HAL_PKA_ECCCompleteAddition_IT()
   – HAL_PKA_ECCCompleteAddition_GetResult();

This subsection provides a set of functions allowing to manage the PKA operations. (#) There are two modes of operation: (++) Blocking mode : The operation is performed in the polling mode. These functions return when data operation is completed. (++) No-Blocking mode : The operation is performed using Interrupts. These functions return immediately. The end of the operation is indicated by HAL_PKA_ErrorCallback in case of error. The end of the operation is indicated by HAL_PKA_OperationCpltCallback in case of success. To stop any operation in interrupt mode, use HAL_PKA_Abort(). (#) Blocking mode functions are :

- Operation for stm32wb05, stm32wb06, stm32wb07, stm32wb09:
  – HAL_PKA_ECCMul()
  – HAL_PKA_ECCMul_GetResult();

- Operation for stm32wb05, stm32wb09 only:
  - HAL_PKA_ModExp()
  - HAL_PKA_ModExpFastMode()
  - HAL_PKA_ModExp_GetResult();
  - HAL_PKA_ECDSASign()
  - HAL_PKA_ECDSASign_GetResult();
  - HAL_PKA_ECDSAVerif()
  - HAL_PKA_ECDSAVerif_IsValidSignature();
  - HAL_PKA_RSACRTExp()
  - HAL_PKA_RSACRTExp_GetResult();
  - HAL_PKA_PointCheck()
  - HAL_PKA_PointCheck_IsOnCurve();
  - HAL_PKA_ECCMulFastMode()
  - HAL_PKA_Add()
  - HAL_PKA_Sub()
  - HAL_PKA_Cmp()
  - HAL_PKA_Mul()
  - HAL_PKA_ModAdd()
  - HAL_PKA_ModSub()
  - HAL_PKA_ModInv()
  - HAL_PKA_ModRed()
  - HAL_PKA_MontgomeryMul()
  - HAL_PKA_Arithmetic_GetResult(P);
  - HAL_PKA_MontgomeryParam()
  - HAL_PKA_MontgomeryParam_GetResult();
- Operation for stm32wb09 only:
  - HAL_PKA_ECCDoubleBaseLadder()
  - HAL_PKA_ECCDoubleBaseLadder_GetResult();
  - HAL_PKA_ECCCompleteAddition()
  - HAL_PKA_ECCCompleteAddition_GetResult(); (#) No-Blocking mode functions with Interrupt are :
- Operation for stm32wb05, stm32wb06, stm32wb07, stm32wb09:
  - HAL_PKA_ECCMul_IT();
  - HAL_PKA_ECCMul_GetResult();
  - HAL_PKA_Abort();

- Operation for stm32wb05, stm32wb09:
  - HAL_PKA_ModExp_IT();
  - HAL_PKA_ModExpFastMode_IT();
  - HAL_PKA_ModExp_GetResult();
  - HAL_PKA_ECDSASign_IT();
  - HAL_PKA_ECDSASign_GetResult();
  - HAL_PKA_ECDSAVerif_IT();
  - HAL_PKA_ECDSAVerif_IsValidSignature();
  - HAL_PKA_RSACRTExp_IT();
  - HAL_PKA_RSACRTExp_GetResult();
  - HAL_PKA_PointCheck_IT();
  - HAL_PKA_PointCheck_IsOnCurve();
  - HAL_PKA_ECCMulFastMode_IT();
  - HAL_PKA_Add_IT();
  - HAL_PKA_Sub_IT();
  - HAL_PKA_Cmp_IT();
  - HAL_PKA_Mul_IT();
  - HAL_PKA_ModAdd_IT();
  - HAL_PKA_ModSub_IT();
  - HAL_PKA_ModInv_IT();
  - HAL_PKA_ModRed_IT();
  - HAL_PKA_MontgomeryMul_IT();
  - HAL_PKA_Arithmetic_GetResult();
  - HAL_PKA_MontgomeryParam_IT();
  - HAL_PKA_MontgomeryParam_GetResult();
- Operation for stm32wb09:
  - HAL_PKA_ECCDoubleBaseLadder_IT()
  - HAL_PKA_ECCDoubleBaseLadder_GetResult();
  - HAL_PKA_ECCCompleteAddition_IT()
  - HAL_PKA_ECCCompleteAddition_GetResult();

This section contains the following APIs:
- *HAL_PKA_ECCMul()*
- *HAL_PKA_ECCMul_IT()*
- *HAL_PKA_ECCMul_GetResult()*
- *HAL_PKA_Abort()*
- *HAL_PKA_RAMReset()*
- *HAL_PKA_IRQHandler()*
- *HAL_PKA_OperationCpltCallback()*
- *HAL_PKA_ErrorCallback()*

22.2.4 Peripheral State and Error functions

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

This section contains the following APIs:
- *HAL_PKA_GetState()*
- *HAL_PKA_GetError()*

22.2.5 Detailed description of functions

HAL_PKA_Init

Function name

HAL_StatusTypeDef HAL_PKA_Init (PKA_HandleTypeDef * hpka)

Function description

Initialize the PKA according to the specified parameters in the PKA_InitTypeDef and initialize the associated handle.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_DeInit

Function name

HAL_StatusTypeDef HAL_PKA_DeInit (PKA_HandleTypeDef * hpka)

Function description

DeInitialize the PKA peripheral.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_MspInit

Function name

void HAL_PKA_MspInit (PKA_HandleTypeDef * hpka)

Function description

Initialize the PKA MSP.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_MspDeInit

Function name

void HAL_PKA_MspDeInit (PKA_HandleTypeDef * hpka)

Function description

DeInitialize the PKA MSP.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_RegisterCallback

Function name

HAL_StatusTypeDef HAL_PKA_RegisterCallback (PKA_HandleTypeDef * hpka, HAL_PKA_CallbackIDTypeDef CallbackID, pPKA_CallbackTypeDef pCallback)

Function description

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

Parameters

- **hpka:** Pointer to a PKA_HandleTypeDef structure that contains the configuration information for the specified PKA.
- **CallbackID:** ID of the callback to be registered This parameter can be one of the following values:
  - HAL_PKA_OPERATION_COMPLETE_CB_ID End of operation callback ID
  - HAL_PKA_ERROR_CB_ID Error callback ID
  - HAL_PKA_MSPINIT_CB_ID MspInit callback ID
  - HAL_PKA_MSPDEINIT_CB_ID MspDeInit callback ID
- **pCallback:** pointer to the Callback function

Return values

- **HAL:** status

HAL_PKA_UnRegisterCallback

Function name

HAL_StatusTypeDef HAL_PKA_UnRegisterCallback (PKA_HandleTypeDef * hpka, HAL_PKA_CallbackIDTypeDef CallbackID)

Function description

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

Parameters

- **hpka:** Pointer to a PKA_HandleTypeDef structure that contains the configuration information for the specified PKA.
- **CallbackID:** ID of the callback to be unregistered This parameter can be one of the following values:
  - HAL_PKA_OPERATION_COMPLETE_CB_ID End of operation callback ID
  - HAL_PKA_ERROR_CB_ID Error callback ID
  - HAL_PKA_MSPINIT_CB_ID MspInit callback ID
  - HAL_PKA_MSPDEINIT_CB_ID MspDeInit callback ID

Return values

- **HAL:** status

HAL_PKA_ECCMul

Function name

HAL_StatusTypeDef HAL_PKA_ECCMul (PKA_HandleTypeDef * hpka, PKA_ECCMulInTypeDef * in, uint32_t Timeout)

Function description

ECC scalar multiplication in blocking mode.

Parameters

- **hpka:** PKA handle
- **in:** Input information
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_PKA_ECCMul_IT

Function name

HAL_StatusTypeDef HAL_PKA_ECCMul_IT (PKA_HandleTypeDef * hpka, PKA_ECCMulInTypeDef * in)

Function description

ECC scalar multiplication in non-blocking mode with Interrupt.

Parameters

- **hpka:** PKA handle
- **in:** Input information

Return values

- **HAL:** status

HAL_PKA_ECCMul_GetResult

Function name

void HAL_PKA_ECCMul_GetResult (PKA_HandleTypeDef * hpka, PKA_ECCMulOutTypeDef * out)

Function description

Retrieve operation result.

Parameters

- **hpka:** PKA handle
- **out:** Output information

Return values

- **HAL:** status

HAL_PKA_Abort

Function name

HAL_StatusTypeDef HAL_PKA_Abort (PKA_HandleTypeDef * hpka)

Function description

Abort any ongoing operation.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_RAMReset

Function name

void HAL_PKA_RAMReset (PKA_HandleTypeDef * hpka)

Function description

Reset the PKA RAM.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_OperationCpltCallback

Function name

void HAL_PKA_OperationCpltCallback (PKA_HandleTypeDef * hpka)

Function description

Process completed callback.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_ErrorCallback

Function name

void HAL_PKA_ErrorCallback (PKA_HandleTypeDef * hpka)

Function description

Error callback.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_IRQHandler

Function name

void HAL_PKA_IRQHandler (PKA_HandleTypeDef * hpka)

Function description

This function handles PKA event interrupt request.

Parameters

- **hpka:** PKA handle

Return values

- **None:**

HAL_PKA_GetState

Function name

HAL_PKA_StateTypeDef HAL_PKA_GetState (const PKA_HandleTypeDef * hpka)

Function description

Return the PKA handle state.

Parameters

- **hpka:** PKA handle

Return values

- **HAL:** status

HAL_PKA_GetError

Function name

uint32_t HAL_PKA_GetError (const PKA_HandleTypeDef * hpka)

Function description

Return the PKA error code.

Parameters

- **hpka:** PKA handle

Return values

- **PKA:** error code

22.3 PKA Firmware driver defines

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

22.3.1 PKA

PKA

PKA Error Code definition

HAL_PKA_ERROR_NONE

HAL_PKA_ERROR_ADDRERR

HAL_PKA_ERROR_RAMERR

HAL_PKA_ERROR_TIMEOUT

HAL_PKA_ERROR_OPERATION

HAL_PKA_ERROR_INVALID_CALLBACK

Invalid Callback error

PKA Exported Macros

__HAL_PKA_RESET_HANDLE_STATE

Description:

- Reset PKA handle state.

Parameters:

- __HANDLE__: specifies the PKA Handle

Return value:

- None

23 HAL PWR Generic Driver

23.1 PWR Firmware driver registers structures

23.1.1 PWR_PVDTypeDef

PWR_PVDTypeDef is defined in the stm32wb0x_hal_pwr.h

Data Fields

- ***uint32_t PVDLevel***
- ***uint32_t Mode***

Field Documentation

- ***uint32_t PWR_PVDTypeDef::PVDLevel***
  PVDLevel: Specifies the PVD detection level. This parameter can be a value of ***PWR_PVD_Detection_Level***.
- ***uint32_t PWR_PVDTypeDef::Mode***
  Mode: Specifies the operating mode for the selected pins. This parameter can be a value of ***PWR_PVD_Mode***.

23.1.2 PWR_DEEPSTOPTypeDef

PWR_DEEPSTOPTypeDef is defined in the stm32wb0x_hal_pwr.h

Data Fields

- ***uint32_t deepStopMode***

Field Documentation

- ***uint32_t PWR_DEEPSTOPTypeDef::deepStopMode***
  Specifies the configuration of the slow clock in DEEPSTOP. This parameter can be one of the following value:
  - PWR_DEEPSTOP_WITH_SLOW_CLOCK_OFF
  - PWR_DEEPSTOP_WITH_SLOW_CLOCK_ON

23.1.3 PWR_SHUTDOWNTypeDef

PWR_SHUTDOWNTypeDef is defined in the stm32wb0x_hal_pwr.h

Data Fields

- ***uint8_t BORStatus***
- ***uint8_t WakeUpPinStatus***
- ***uint8_t WakeUpPol***

Field Documentation

- ***uint8_t PWR_SHUTDOWNTypeDef::BORStatus***
  Specifies the BOR status ENABLE or DISABLE
- ***uint8_t PWR_SHUTDOWNTypeDef::WakeUpPinStatus***
  Specifies if the PB0 wake up source is ENABLE or DISABLE
- ***uint8_t PWR_SHUTDOWNTypeDef::WakeUpPol***
  Specifies the wake up source polarity

23.2 PWR Firmware driver API description

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

23.2.1 Initialization and De-Initialization Functions

This section provides functions allowing to deinitialize power peripheral.

This section contains the following APIs:

- ***HAL_PWR_DeInit()***

23.2.2 Peripheral Control functions

- The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller.
- Every entity has low power mode as described below:
  - The CPU low power modes are : (+) CPU CRun. (+) CPU Sleep WFI. (+) CPU DeepSleep.
  - The Core low power modes are : (+) Run. (+) Sleep. (+) DEEPSTOP with with retention and low speed clock enabled. (+) DEEPSTOP with with retention and low speed clock disabled. (+) Shutdown.

(#) The Power control (PWR) provides an overview of the supply architecture for the different power domains and of the supply configuration controller. (#) Every entity has low power mode as described below : (#) The CPU low power modes are :

- CPU CRun.
- CPU Sleep WFI.
- CPU DeepSleep. (#) The Core low power modes are :
- Run.
- Sleep.
- DEEPSTOP with with retention and low speed clock enabled.
- DEEPSTOP with with retention and low speed clock disabled.
- Shutdown.

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PVDLS[2:0] bits in PWR_CR2 register).
- PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event can generate an interrupt if enabled. This is done through __HAL_PVD_ENABLE_IT() macro.
- The PVD is stopped in Shutdown mode.

WakeUp pin configuration

- Wake-up pin is used to wake up the system from DEEPSTOP mode. The pin selection is configurable through the CR3 and CR6 registers to map signal to wake up pin line. The pin polarity is configurable through the CR4 and CR7 registers to be active on rising or falling edges.
- When a wakeup pin event is received the appropriate flag is set in the SR1 and SR3 registers. Then the wakeup pin flag will be cleared and the IOs user callback will be called. The user can add his own code by customization of this function HAL_PWR_WKUPx_Callback.

This section contains the following APIs:

- *HAL_PWR_ConfigPVD()*
- *HAL_PWR_EnablePVD()*
- *HAL_PWR_DisablePVD()*
- *HAL_PWR_EnableWakeUpPin()*
- *HAL_PWR_DisableWakeUpPin()*
- *HAL_PWR_GetClearWakeupSource()*
- *HAL_PWR_EnterSLEEPMode()*
- *HAL_PWR_ConfigDEEPSTOP()*
- *HAL_PWR_EnterDEEPSTOPMode()*
- *HAL_PWR_ConfigSHUTDOWN()*
- *HAL_PWREx_EnterSHUTDOWNMode()*
- *HAL_PWR_EnableSleepOnExit()*
- *HAL_PWR_DisableSleepOnExit()*
- *HAL_PWR_EnableSEVOnPend()*
- *HAL_PWR_DisableSEVOnPend()*
- *HAL_PWR_PVD_IRQHandler()*

- *HAL_PWR_PVDCallback()*
- *HAL_PWR_WKUP_IRQHandler()*
- *HAL_PWR_WKUPx_Callback()*

23.2.3 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.:**

Notes

- This functionality is not available in this product. The prototype is kept just to maintain compatibility with other products.

HAL_PWR_ConfigPVD

Function name

HAL_StatusTypeDef HAL_PWR_ConfigPVD (const PWR_PVDTypeDef * sConfigPVD)

Function description

Configure the voltage threshold detected by the Power Voltage Detector (PVD).

Parameters

- **sConfigPVD:** pointer to a PWR_PVDTypeDef structure that contains the PVD configuration information.

Return values

- **None:**

Notes

- Refer to the electrical characteristics of your device datasheet for more details about the voltage thresholds corresponding to each detection level.

HAL_PWR_EnablePVD

Function name

void HAL_PWR_EnablePVD (void)

Function description

Enables the Power Voltage Detector(PVD).

Return values

- **None:**

HAL_PWR_DisablePVD

Function name

void HAL_PWR_DisablePVD (void)

Function description

Disables the Power Voltage Detector(PVD).

Return values

- **None:**

HAL_PWR_PVDCallback

Function name

void HAL_PWR_PVDCallback (void)

Function description

PWR PVD interrupt callback.

Return values

- **None:**

HAL_PWR_EnableWakeUpPin

Function name

void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPin, uint32_t WakeUpPolarity)

Function description

Enable the WakeUp PINx functionality.

Parameters

- **WakeUpPin:** Specifies which Wake-Up pin to enable. This parameter can be one of the following:
  - PWR_WAKEUP_PA0
  - PWR_WAKEUP_PA1
  - PWR_WAKEUP_PA2
  - PWR_WAKEUP_PA3
  - PWR_WAKEUP_PA4 (*)
  - PWR_WAKEUP_PA5 (*)
  - PWR_WAKEUP_PA6 (*)
  - PWR_WAKEUP_PA7 (*)
  - PWR_WAKEUP_PA8
  - PWR_WAKEUP_PA9
  - PWR_WAKEUP_PA10
  - PWR_WAKEUP_PA11
  - PWR_WAKEUP_PA12 (*)
  - PWR_WAKEUP_PA13 (*)
  - PWR_WAKEUP_PA14 (*)
  - PWR_WAKEUP_PA15 (*)
  - PWR_WAKEUP_PB0
  - PWR_WAKEUP_PB1
  - PWR_WAKEUP_PB2
  - PWR_WAKEUP_PB3
  - PWR_WAKEUP_PB4
  - PWR_WAKEUP_PB5
  - PWR_WAKEUP_PB6
  - PWR_WAKEUP_PB7
  - PWR_WAKEUP_PB8 (*)
  - PWR_WAKEUP_PB9 (*)
  - PWR_WAKEUP_PB10 (*)
  - PWR_WAKEUP_PB11 (*)
  - PWR_WAKEUP_PB12 (**)
  - PWR_WAKEUP_PB13 (**)
  - PWR_WAKEUP_PB14 (**)
  - PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices
- **WakeUpPolarity:** Specifies the polarity of the wake up pin source. This parameter can be one of the following:
  - PWR_WUP_RISIEDG
  - PWR_WUP_FALLEDG

Return values

- **None:**

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:
  - PWR_WAKEUP_PA0
  - PWR_WAKEUP_PA1
  - PWR_WAKEUP_PA2
  - PWR_WAKEUP_PA3
  - PWR_WAKEUP_PA4 (*)
  - PWR_WAKEUP_PA5 (*)
  - PWR_WAKEUP_PA6 (*)
  - PWR_WAKEUP_PA7 (*)
  - PWR_WAKEUP_PA8
  - PWR_WAKEUP_PA9
  - PWR_WAKEUP_PA10
  - PWR_WAKEUP_PA11
  - PWR_WAKEUP_PA12 (*)
  - PWR_WAKEUP_PA13 (*)
  - PWR_WAKEUP_PA14 (*)
  - PWR_WAKEUP_PA15 (*)
  - PWR_WAKEUP_PB0
  - PWR_WAKEUP_PB1
  - PWR_WAKEUP_PB2
  - PWR_WAKEUP_PB3
  - PWR_WAKEUP_PB4
  - PWR_WAKEUP_PB5
  - PWR_WAKEUP_PB6
  - PWR_WAKEUP_PB7
  - PWR_WAKEUP_PB8 (*)
  - PWR_WAKEUP_PB9 (*)
  - PWR_WAKEUP_PB10 (*)
  - PWR_WAKEUP_PB11 (*)
  - PWR_WAKEUP_PB12 (**)
  - PWR_WAKEUP_PB13 (**)
  - PWR_WAKEUP_PB14 (**)
  - PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

- **None:**

HAL_PWR_GetClearWakeupSource

Function name

uint32_t HAL_PWR_GetClearWakeupSource (void)

Function description

Get and Clear Wake-up source.

Return values

- **WakeUpPin:** : This parameter can be a combination of PWR Wake-up pins define group

HAL_PWR_ConfigDEEPSTOP

Function name

HAL_StatusTypeDef HAL_PWR_ConfigDEEPSTOP (PWR_DEEPSTOPTypeDef * sConfigDEEPSTOP)

Function description

Configures the system to allow the DEEPSTOP mode.

Parameters

- **sConfigDEEPSTOP:** : Pointer to a PWR_DEEPSTOPTypeDef structure that contains the DEEPSTOP configuration information.

Return values

- **None.:**

HAL_PWR_ConfigSHUTDOWN

Function name

HAL_StatusTypeDef HAL_PWR_ConfigSHUTDOWN (PWR_SHUTDOWNTypeDef * sConfigSHUTDOWN)

Function description

Configures the system to allow the SHUTDOWN mode.

Parameters

- **sConfigSHUTDOWN:** : Pointer to a PWR_SHUTDOWNTypeDef structure that contains the SHUTDOWN configuration information.

Return values

- **None.:**

HAL_PWR_EnterDEEPSTOPMode

Function name

void HAL_PWR_EnterDEEPSTOPMode (void)

Function description

Enter DEEPSTOP mode.

Return values

- **None:**

Notes

- The DEEPSTOP is the only low power mode of the device allowing to restart from a saved context environment and go on running the application at wakeup.
- CPU clock and bus clocks are stopped, the VDD12i power domain is switched off, the VDD12o power domain is ON and supplied at 1.0V. All the RAM banks are retained (by configuration in SystemInit()). The slow clock can be running or stopped, depending on the software configuration. All the register content is lost. Only few peripheral can save the setting if configured like wake up source.

HAL_PWREx_EnterSHUTDOWNMode

Function name

void HAL_PWREx_EnterSHUTDOWNMode() (void)

Function description

Enter SHUTDOWN mode.

Return values

- **None.:**

Notes

- In Shutdown mode, all clocks are off as RC64MPLL, LSI and LSE are switched off. The system is powered down as both regulators are OFF is powered off.
- 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 Pull Down 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_EnterSLEEPMode

Function name

void HAL_PWR_EnterSLEEPMode (void)

Function description

Enter the CPU in Sleep mode.

Return values

- **None.:**

Notes

- In Sleep mode, all I/O pins keep the same state as in Run mode.
- CPU clock is off and all peripherals including Cortex-M0+ core such as NVIC and SysTick can run and wake up the CPU when an interrupt or an event occurs.

HAL_PWR_EnableSleepOnExit

Function name

void HAL_PWR_EnableSleepOnExit (void)

Function description

Indicate SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters Sleep mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

HAL_PWR_DisableSleepOnExit

Function name

void HAL_PWR_DisableSleepOnExit (void)

Function description

Disable SLEEP-ON-EXIT feature when returning from handler mode to thread mode.

Return values

- **None.:**

Notes

- Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters Sleep mode when an interruption handling is over.

HAL_PWR_EnableSEVOnPend

Function name

void HAL_PWR_EnableSEVOnPend (void)

Function description

Enable Cortex Sev On 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

HAL_PWR_PVD_IRQHandler

Function name

void HAL_PWR_PVD_IRQHandler (void)

Function description

This function handles the PWR PVD interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the PVD_IRQHandler().

HAL_PWR_WKUP_IRQHandler

Function name

void HAL_PWR_WKUP_IRQHandler (void)

Function description

This function handles the PWR WAKEUP interrupt request.

Return values

- **None.:**

Notes

- This API should be called under the WKUP_IRQHandler().

HAL_PWR_WKUPx_Callback

Function name

void HAL_PWR_WKUPx_Callback (uint32_t WakeupIOs)

Function description

PWR WKUPx interrupt callback.

Parameters

- **WakeupIOs:** IO wakeup line

Return values

- **None.:**

23.3 PWR Firmware driver defines

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

23.3.1 PWR

PWR

Slow clock configuration in DEEPSTOP mode

PWR_DEEPSTOP_WITH_SLOW_CLOCK_OFF

PWR_DEEPSTOP_WITH_SLOW_CLOCK_ON

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 of the following values:

Return value:

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

__HAL_PWR_CLEAR_FLAG

Description:

- Clear a specific PWR flag.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
  - PWR_FLAG_WUF0 Wake Up Flag 0. Indicates that a wakeup event was received from the pin PB0.
  - PWR_FLAG_WUF1 Wake Up Flag 1. Indicates that a wakeup event was received from the pin PB1.
  - PWR_FLAG_WUF2 Wake Up Flag 2. Indicates that a wakeup event was received from the pin PB2.
  - PWR_FLAG_WUF3 Wake Up Flag 3. Indicates that a wakeup event was received from the pin PB3.
  - PWR_FLAG_WUF4 Wake Up Flag 4. Indicates that a wakeup event was received from the pin PB4.
  - PWR_FLAG_WUF5 Wake Up Flag 5. Indicates that a wakeup event was received from the pin PB5.
  - PWR_FLAG_WUF6 Wake Up Flag 6. Indicates that a wakeup event was received from the pin PB6.
  - PWR_FLAG_WUF7 Wake Up Flag 7. Indicates that a wakeup event was received from the pin PB7.
  - PWR_FLAG_WUF8 Wake Up Flag 8. Indicates that a wakeup event was received from the pin PA8.
  - PWR_FLAG_WUF9 Wake Up Flag 9. Indicates that a wakeup event was received from the pin PA9.
  - PWR_FLAG_WUF10 Wake Up Flag 10. Indicates that a wakeup event was received from the pin PA10.
  - PWR_FLAG_WUF11 Wake Up Flag 11. Indicates that a wakeup event was received from the pin PA11.
  - PWR_FLAG_WBLEF Wake Up Flag Indicates that a wakeup event was received from the Bluetooth LE.
  - PWR_FLAG_BHWF Wake Up Flag Indicates that a wakeup event was received from the Bluetooth LE Host CPU.
  - PWR_FLAG_IWUF2 Wake Up Flag 11. Indicates that a wakeup event was received from the pin PA11.
  - PWR_FLAG_WUF12 Wake Up Flag 12. Indicates that a wakeup event was received from the pin PA0.
  - PWR_FLAG_WUF13 Wake Up Flag 13. Indicates that a wakeup event was received from the pin PA1.
  - PWR_FLAG_WUF14 Wake Up Flag 14. Indicates that a wakeup event was received from the pin PA2.
  - PWR_FLAG_WUF15 Wake Up Flag 15. Indicates that a wakeup event was received from the pin PA3.
  - PWR_FLAG_WUF16 Wake Up Flag 16. Indicates that a wakeup event was received from the pin PB12 or PA4 for STM32wb07.
  - PWR_FLAG_WUF17 Wake Up Flag 17. Indicates that a wakeup event was received from the pin PB13 or PA5 for STM32wb07.
  - PWR_FLAG_WUF18 Wake Up Flag 18. Indicates that a wakeup event was received from the pin PB14 or PA6 for STM32wb07.
  - PWR_FLAG_WUF19 Wake Up Flag 19. Indicates that a wakeup event was received from the pin PB15 or PA7 for STM32wb07.
  - PWR_FLAG_WUF20 Wake Up Flag 20. Indicates that a wakeup event was received from the pin PB8.
  - PWR_FLAG_WUF21 Wake Up Flag 21. Indicates that a wakeup event was received from the pin PB9.
  - PWR_FLAG_WUF22 Wake Up Flag 22. Indicates that a wakeup event was received from the pin PB10.
  - PWR_FLAG_WUF23 Wake Up Flag 23. Indicates that a wakeup event was received from the pin PB11.
  - PWR_FLAG_WUF24 Wake Up Flag 24. Indicates that a wakeup event was received from the pin PB12.
  - PWR_FLAG_WUF25 Wake Up Flag 25. Indicates that a wakeup event was received from the pin PB13.
  - PWR_FLAG_WUF26 Wake Up Flag 26. Indicates that a wakeup event was received from the pin PB14.
  - PWR_FLAG_WUF27 Wake Up Flag 27. Indicates that a wakeup event was received from the pin PB15.

- PWR_WU_FLAG_ALL all Wakeup flags.
- PWR_FLAG_DEEPSTOPF System DEEPTSTOP Flag
- PWR_FLAG_RFPHASEF RFPHASE Flag

Return value:

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

__HAL_PWR_PVD_ENABLE_IT

Description:

- Enable the PVD Line.

Return value:

- None

__HAL_PWR_PVD_DISABLE_IT

Description:

- Disable the PVD Interrupt Line.

Return value:

- None

__HAL_PWR_PVD_GET_FLAG

Description:

- Check whether or not the PVD interrupt flag is set.

Return value:

- PVD: Line Status.

__HAL_PWR_PVD_CLEAR_FLAG

Description:

- Clear the PVD interrupt flag.

Return value:

- None

PWR Low Power Mode Selection

PWR_LOWPOWERMODE_STOP

DeepStop mode

PWR_LOWPOWERMODE_SHUTDOWN

Shutdown mode

Power Voltage Detector Level selection

PWR_PVDLEVEL_0

PVD threshold around 2.0 V

PWR_PVDLEVEL_1

PVD threshold around 2.2 V

PWR_PVDLEVEL_2

PVD threshold around 2.4 V

PWR_PVDLEVEL_3

PVD threshold around 2.5 V

PWR_PVDLEVEL_4

PVD threshold around 2.6 V

PWR_PVDLEVEL_5

PVD threshold around 2.8 V

PWR_PVDLEVEL_6

PVD threshold around 2.9 V

PWR_PVDLEVEL_7

External input analog voltage (Compare internally to VBGP)

PWR PVD interrupt and event mode

PWR_PVD_MODE_NORMAL

Basic mode is used

PWR_PVD_MODE_IT

Interrupt Mode detection

RAM Retention enable

PWR_RAMRET_1

PWR_RAMRET_2

PWR_RAMRET_3

PWR regulator mode

PWR_MAINREGULATOR_ON

Regulator in main mode

PWR_LOWPOWERREGULATOR_ON

Regulator in low-power mode

24 HAL PWR Extension Driver

24.1 PWREx Firmware driver defines

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

24.1.1 PWREx

PWREx

PWR Status Flags

PWR_FLAG_WUF0

Wakeup event on wakeup pin PB0

PWR_FLAG_WUF1

Wakeup event on wakeup pin PB1

PWR_FLAG_WUF2

Wakeup event on wakeup pin PB2

PWR_FLAG_WUF3

Wakeup event on wakeup pin PB3

PWR_FLAG_WUF4

Wakeup event on wakeup pin PB4

PWR_FLAG_WUF5

Wakeup event on wakeup pin PB5

PWR_FLAG_WUF6

Wakeup event on wakeup pin PB6

PWR_FLAG_WUF7

Wakeup event on wakeup pin PB7

PWR_FLAG_WUF8

Wakeup event on wakeup pin PA8

PWR_FLAG_WUF9

Wakeup event on wakeup pin PA9

PWR_FLAG_WUF10

Wakeup event on wakeup pin PA10

PWR_FLAG_WUF11

Wakeup event on wakeup pin PA11

PWR_FLAG_WBLEF

Bluetooth LE WakeUp Flag

PWR_FLAG_BHWF

Bluetooth LE Host CPU WakeUp Flag

PWR_FLAG_WUFI

Wakeup on internal wakeup line

PWR_FLAG_IWUF2

Wakeup on internal wakeup 2 line

PWR_FLAG_SR1_ALL

PWR_FLAG_SMPSBYPR

SMPS in SMPS PRECHARGE mode

PWR_FLAG_SMPSENR

SMPS in RUN mode Flag

PWR_FLAG_SMPSRDYF

SMPS Ready Flag

PWR_FLAG_REGLPS

Low-power regulator start flag

PWR_FLAG_PVDO

Power Voltage Detector output flag

PWR_FLAG_WUF12

Wakeup event on wakeup pin PA0

PWR_FLAG_WUF13

Wakeup event on wakeup pin PA1

PWR_FLAG_WUF14

Wakeup event on wakeup pin PA2

PWR_FLAG_WUF15

Wakeup event on wakeup pin PA3

PWR_FLAG_WUF16

Wakeup event on wakeup pin PB12

PWR_FLAG_WUF17

Wakeup event on wakeup pin PB13

PWR_FLAG_WUF18

Wakeup event on wakeup pin PB14

PWR_FLAG_WUF19

Wakeup event on wakeup pin PB15

PWR_FLAG_SR3_ALL

PWR_WU_FLAG_ALL

PWR_FLAG_DEEPSTOPF

System DEEPTSTOP Flag

PWR_FLAG_RFPHASEF

RFPHASE Flag

PWR status register

PWR_FLAG_REG_SR1

Bitfield to indicate PWR flag located in register PWR_SR1

PWR_FLAG_REG_SR2

Bitfield to indicate PWR flag located in register PWR_SR2

PWR_FLAG_REG_SR3

Bitfield to indicate PWR flag located in register PWR_SR3

PWR_FLAG_REG_EXTSRR

Bitfield to indicate PWR flag located in register PWR_EXTSRR

PWR_FLAG_REG_MASK

Bitfield mask to indicate PWR flag location in PWR register

GPIO port

PWR_GPIO_A

GPIO port A

PWR_GPIO_B

GPIO port B

GPIO bit number I/O setting in shutdown mode

PWR_GPIO_BIT_0

GPIO port I/O pin 0

PWR_GPIO_BIT_1

GPIO port I/O pin 1

PWR_GPIO_BIT_2

GPIO port I/O pin 2

PWR_GPIO_BIT_3

GPIO port I/O pin 3

PWR_GPIO_BIT_4

GPIO port I/O pin 4

PWR_GPIO_BIT_5

GPIO port I/O pin 5

PWR_GPIO_BIT_6

GPIO port I/O pin 6

PWR_GPIO_BIT_7

GPIO port I/O pin 7

PWR_GPIO_BIT_8

GPIO port I/O pin 8

PWR_GPIO_BIT_9

GPIO port I/O pin 9

PWR_GPIO_BIT_10

GPIO port I/O pin 10

PWR_GPIO_BIT_11

GPIO port I/O pin 11

PWR_GPIO_BIT_12

GPIO port I/O pin 12

PWR_GPIO_BIT_13

GPIO port I/O pin 13

PWR_GPIO_BIT_14

GPIO port I/O pin 14

PWR_GPIO_BIT_15

GPIO port I/O pin 15

PWR Extended GPIO Pull-Down Port

PWR_GPIO_PULLDOWN_PORT_A

GPIO port A

PWR_GPIO_PULLDOWN_PORT_B

GPIO port B

PWR Extended GPIO Pull-Up Port

PWR_GPIO_PULLUP_PORT_A

GPIO port A

PWR_GPIO_PULLUP_PORT_B

GPIO port B

PWREx Pin Polarity configuration

PWR_WUP_RISIEDG

PWR_WUP_FALLEDG

PWREx Pin Pull configuration

PWR_PIN_NO_PULL

PWR_PIN_PULL_UP

PWR_PIN_PULL_DOWN

SMPS step down converter operating modes

PWR_SMPS_ON

SMPS step down ON

PWR_SMPS_OFF

SMPS step down OFF

PWR_SMPS_BYPASS

SMPS is disabled and bypassed in precharge mode

SMPS step down converter output voltage scaling voltage level

PWR_SMPS_OUTPUT_VOLTAGE_1V20

SMPS step down converter supply output voltage 1.20V

PWR_SMPS_OUTPUT_VOLTAGE_1V25
SMPS step down converter supply output voltage 1.25V

PWR_SMPS_OUTPUT_VOLTAGE_1V30
SMPS step down converter supply output voltage 1.30V

PWR_SMPS_OUTPUT_VOLTAGE_1V35
SMPS step down converter supply output voltage 1.35V

PWR_SMPS_OUTPUT_VOLTAGE_1V40
SMPS step down converter supply output voltage 1.40V

PWR_SMPS_OUTPUT_VOLTAGE_1V45
SMPS step down converter supply output voltage 1.45V

PWR_SMPS_OUTPUT_VOLTAGE_1V50
SMPS step down converter supply output voltage 1.50V

PWR_SMPS_OUTPUT_VOLTAGE_1V55
SMPS step down converter supply output voltage 1.55V

PWR_SMPS_OUTPUT_VOLTAGE_1V60
SMPS step down converter supply output voltage 1.60V

PWR_SMPS_OUTPUT_VOLTAGE_1V65
SMPS step down converter supply output voltage 1.65V

PWR_SMPS_OUTPUT_VOLTAGE_1V70
SMPS step down converter supply output voltage 1.70V

PWR_SMPS_OUTPUT_VOLTAGE_1V75
SMPS step down converter supply output voltage 1.75V

PWR_SMPS_OUTPUT_VOLTAGE_1V80
SMPS step down converter supply output voltage 1.80V

PWR_SMPS_OUTPUT_VOLTAGE_1V85
SMPS step down converter supply output voltage 1.85V

PWR_SMPS_OUTPUT_VOLTAGE_1V90
SMPS step down converter supply output voltage 1.90V

PWR_SMPS_OUTPUT_VOLTAGE_1V95
SMPS step down converter supply output voltage 1.95V

PWR Wake-up pins

PWR_WAKEUP_CR3_ALL

PWR_WAKEUP_CR6_ALL

PWR_WAKEUP_ALL
Enable all the wakeup source

PWR_WAKEUP_PIN_RTC
Enable wakeup on Internal event (RTC)

PWR_WAKEUP_PIN_LPUART

Enable wakeup on Internal event 2 (LPUART)

PWR_WAKEUP_BLEHOST

Enable wakeup on BLE Host CPU event

PWR_WAKEUP_BLE

Enable wakeup on BLE event

PWR_WAKEUP_PA11

Enable wakeup on PA11 I/O event

PWR_WAKEUP_PA10

Enable wakeup on PA10 I/O event

PWR_WAKEUP_PA9

Enable wakeup on PA9 I/O event

PWR_WAKEUP_PA8

Enable wakeup on PA8 I/O event

PWR_WAKEUP_PA3

Enable wakeup on PA3 I/O event

PWR_WAKEUP_PA2

Enable wakeup on PA2 I/O event

PWR_WAKEUP_PA1

Enable wakeup on PA1 I/O event

PWR_WAKEUP_PA0

Enable wakeup on PA0 I/O event

PWR_WAKEUP_PB7

Enable wakeup on PB7 I/O event

PWR_WAKEUP_PB6

Enable wakeup on PB6 I/O event

PWR_WAKEUP_PB5

Enable wakeup on PB5 I/O event

PWR_WAKEUP_PB4

Enable wakeup on PB4 I/O event

PWR_WAKEUP_PB3

Enable wakeup on PB3 I/O event

PWR_WAKEUP_PB2

Enable wakeup on PB2 I/O event

PWR_WAKEUP_PB1

Enable wakeup on PB1 I/O event

PWR_WAKEUP_PB0

Enable wakeup on PB0 I/O event

PWR_WAKEUP_PB15

Enable wakeup on PB15 I/O event

PWR_WAKEUP_PB14

Enable wakeup on PB14 I/O event

PWR_WAKEUP_PB13

Enable wakeup on PB13 I/O event

PWR_WAKEUP_PB12

Enable wakeup on PB12 I/O event

Wakeup Source Select

PWR_WUP_CR3

PWR_WUP_CR6

Shift to apply to retrieve polarity information from PWR_WAKEUP_PINy_xxx constants

PWR_WUP_POLARITY_SHIFT

Internal constant used to retrieve wakeup pin polarity

25 HAL RADIO Generic Driver

25.1 RADIO Firmware driver registers structures

25.1.1 RADIO_HandleTypeDef

RADIO_HandleTypeDef is defined in the stm32wb0x_hal_radio.h

Data Fields

- ***BLUE_TypeDef * Instance***

Field Documentation

- ***BLUE_TypeDef* RADIO_HandleTypeDef::Instance***
  Register base address

25.1.2 ActionPacket

ActionPacket is defined in the stm32wb0x_hal_radio.h

Data Fields

- ***uint8_t StateMachineNo***
- ***uint8_t ActionTag***
- ***uint8_t MaxReceiveLength***
- ***uint32_t WakeupTime***
- ***uint8_t * data***
- ***uint32_t status***
- ***uint32_t timestamp_receive***
- ***int32_t rssi***
- ***BlueTransStruct trans_packet***
- ***ActionPacket * next_true***
- ***ActionPacket * next_false***
- ***uint8_t(* condRoutine***
- ***uint8_t(* dataRoutine***
- ***uint8_t trans_config***

Field Documentation

- ***uint8_t ActionPacket::StateMachineNo***
- ***uint8_t ActionPacket::ActionTag***
- ***uint8_t ActionPacket::MaxReceiveLength***
- ***uint32_t ActionPacket::WakeupTime***
- ***uint8_t* ActionPacket::data***
- ***uint32_t ActionPacket::status***
- ***uint32_t ActionPacket::timestamp_receive***
- ***int32_t ActionPacket::rssi***
- ***BlueTransStruct ActionPacket::trans_packet***
- ***ActionPacket* ActionPacket::next_true***
- ***ActionPacket* ActionPacket::next_false***
- ***uint8_t(* ActionPacket::condRoutine)(ActionPacket *)***
- ***uint8_t(* ActionPacket::dataRoutine)(ActionPacket *, ActionPacket *)***
- ***uint8_t ActionPacket::trans_config***

25.1.3 RadioGlobalParameters_t

RadioGlobalParameters_t is defined in the stm32wb0x_hal_radio.h

Data Fields

- *uint32_t back2backTime*
- *uint8_t tone_start_stop_flag*
- *ActionPacket * current_action_packet*

Field Documentation

- *uint32_t RadioGlobalParameters_t::back2backTime*
- *uint8_t RadioGlobalParameters_t::tone_start_stop_flag*
- *ActionPacket* RadioGlobalParameters_t::current_action_packet*

25.1.4 RxStats_t

RxStats_t is defined in the stm32wb0x_hal_radio.h

Data Fields

- *int32_t rssi*
- *uint32_t timestamp_receive*

Field Documentation

- *int32_t RxStats_t::rssi*
- *uint32_t RxStats_t::timestamp_receive*

25.2 RADIO Firmware driver API description

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

25.2.1 Detailed description of functions

HAL_RADIO_Init

Function name

void HAL_RADIO_Init (RADIO_HandleTypeDef * hradio)

Function description

Initializes the radio.

Return values

- **None:**

HAL_RADIO_TXRX_IRQHandler

Function name

void HAL_RADIO_TXRX_IRQHandler (void)

Function description

HAL_RADIO_TXRX_SEQ_IRQHandler

Function name

void HAL_RADIO_TXRX_SEQ_IRQHandler (void)

Function description

HAL_RADIO_TxRxCallback

Function name

void HAL_RADIO_TxRxCallback (uint32_t flags)

Function description

HAL_RADIO_TxRxSeqCallback

Function name

void HAL_RADIO_TxRxSeqCallback (void)

Function description

HAL_RADIO_ReadRSSI

Function name

int8_t HAL_RADIO_ReadRSSI (void)

Function description

Read RSSI.

Return values

- **int8_t:** RSSI in dBm

HAL_RADIO_MspInit

Function name

void HAL_RADIO_MspInit (RADIO_HandleTypeDef * hradio)

Function description

RADIO MSP Init.

Parameters

- **hradio:** pointer to a RADIO_HandleTypeDef structure that contains the configuration information for RADIO module

Return values

- **None:**

HAL_RADIO_MspDeInit

Function name

void HAL_RADIO_MspDeInit (RADIO_HandleTypeDef * hradio)

Function description

RADIO MSP DeInit.

Parameters

- **hradio:** pointer to a RADIO_HandleTypeDef structure that contains the configuration information for RADIO module

Return values

- **None:**

HAL_RADIO_GetStatus

Function name

uint8_t HAL_RADIO_GetStatus (uint32_t * time)

Function description

Get the status of the radio and if it is active the last programmed value in MTU.

Parameters

- **time:** where to store the last value programmed.

Return values

- **0:** if the radio controller is IDLE or no timer has been programmed
- **1:** if the radio is ACTIVE and a radio timer has been programmed

HAL_RADIO_SetChannelMap

Function name

void HAL_RADIO_SetChannelMap (uint8_t StateMachineNo, uint8_t * chan_remap)

Function description

This routine sets the channel map.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **chan_remap:** a 37-bit vector, one bit per channel from 0 to 36, with the LSB corresponding to the channel 0

Return values

- **None:**

HAL_RADIO_SetChannel

Function name

void HAL_RADIO_SetChannel (uint8_t StateMachineNo, uint8_t channel, uint8_t channel_increment)

Function description

This routine sets the channel and the channel increment.

HAL_RADIO_SetTxAttributes

Function name

void HAL_RADIO_SetTxAttributes (uint8_t StateMachineNo, uint32_t NetworkID, uint32_t crc_init)

Function description

This routine sets the NetworkID and the CRC init.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **NetworkID:** The NetworkID is the ID of the device. The user shall ensure that the NetworkID meets the following requirements:
  – It shall have no more than six consecutive zeros or ones.
  – It shall not have all four octets equal.
  – It shall have no more than 24 transitions.
  – It shall have a minimum of two transitions in the most significant six bits.
- **crc_init:** CRC initialization value. This value must be 0x555555 when channel frequency is one of these values: 37, 38, 39.

Return values

- **None:**

HAL_RADIO_SetMaxReceivedLength

Function name

void HAL_RADIO_SetMaxReceivedLength (uint8_t StateMachineNo, uint8_t MaxReceivedLength)

Function description

Set the maximum length of a received packet.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **MaxReceivedLength:** Upper limit for the length of a received packet. From 0 to 255 bytes.

Return values

- **None:**

HAL_RADIO_SetBackToBackTime

Function name

void HAL_RADIO_SetBackToBackTime (uint32_t back_to_back_time)

Function description

Initializes the time between back-to-back radio transmissions.

Parameters

- **back_to_back_time:** time between two packets if wakeupTimer is not used. Resolution is 1 us.

Return values

- **None:**

HAL_RADIO_SetPhy

Function name

void HAL_RADIO_SetPhy (uint8_t StateMachineNo, uint8_t phy)

Function description

Phy selection.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **phy:** 0x0 PHY is not coded 1Mbps 0x1 PHY is not coded 2Mbps 0x4 PHY is coded 1Mbps with S=8 0x6 PHY is coded 1Mbps with S=2

Return values

- **None:**

HAL_RADIO_SetTxPower

Function name

void HAL_RADIO_SetTxPower (uint8_t PowerLevel)

Function description

Configures the transmit power level.

Parameters

- **PowerLevel:** power level which should set to this value. See the documentation inside the datasheet.

Return values

- **None:**

HAL_RADIO_StopActivity

Function name

uint8_t HAL_RADIO_StopActivity (void)

Function description

This routine force the radio to be stopped.

Return values

- **0:** if the radio activity has been cleared successfully.
- **1:** if it is too late to clear the last radio activity.
- **2:** if it might not be possible to clear the last radio activity.

HAL_RADIO_SetGlobalReceiveTimeout

Function name

void HAL_RADIO_SetGlobalReceiveTimeout (uint32_t ReceiveTimeout)

Function description

Set the receive window length.

Parameters

- **ReceiveTimeout:** receive window length in microseconds.

Return values

- **None:**

HAL_RADIO_SetReservedArea

Function name

void HAL_RADIO_SetReservedArea (ActionPacket * p)

Function description

This routine should be called after writing/modifying an action packet, and before it is executed via either the API mechanism, or the next mechanism.

Parameters

- **p:** pointer to action packet.

Return values

- **None:**

HAL_RADIO_MakeActionPacketPending

Function name

uint8_t HAL_RADIO_MakeActionPacketPending (ActionPacket * p)

Function description

This routine should be called for the first actionPacket that needs to be scheduled on the radio.

Parameters

- **p:** pointer to action packet.

Return values

- **uint8_t:** with following values:
  – 0x00 : Success.
  – 0xC4 : Radio is busy, action packet has not been executed.

HAL_RADIO_StartTone

Function name

uint8_t HAL_RADIO_StartTone (uint8_t RF_Channel, uint8_t powerLevel, uint8_t freq_offset)

Function description

Starts tone transmission on selected channel.

Parameters

- **RF_Channel:** radio frequency channel number from 0 to 39. Note that RF channel index is different from ble channel index due to advertising channels, Please refer following table: RF channel 0: 2402 MHz -> ble channel index 37 RF channel 1: 2404 MHz -> ble channel index 0 RF channel 2: 2406 MHz -> ble channel index 1 ... RF channel 11: 2424 MHz -> ble channel index 10 RF channel 12: 2426 MHz -> ble channel index 38 RF channel 13: 2428 MHz -> ble channel index 11 RF channel 14: 2430 MHz -> ble channel index 12 ... RF channel 38: 2478 MHz -> ble channel index 36 RF channel 39: 2480 MHz -> ble channel index 39
- **powerLevel:** power level which should set.
- **freq_offset:** Specify if the tone must be emitted with an offset from the channel center frequency. If 0, the tone is emitted at the channel center frequency. If 1 or 2, the device will continuously emit the tone at the center frequency plus or minus 250 kHz respectively..

Return values

- **None:**

HAL_RADIO_StopTone

Function name

uint8_t HAL_RADIO_StopTone (void)

Function description

This routine stop tone transmission.

Return values

- **None:**

HAL_RADIO_SetEncryptionCount

Function name

void HAL_RADIO_SetEncryptionCount (uint8_t StateMachineNo, uint8_t * count_tx, uint8_t * count_rcv)

Function description

This routine sets the 40-bit receive and transmit packet count, to be used in encryption calculation.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **count_tx:** 40-bit transmit packet count, to be used in encryption nounce calculation.
- **count_rcv:** 40-bit receive packet count, to be used in encryption nounce calculation.

Return values

- **None:**

HAL_RADIO_SetEncryptionAttributes

Function name

void HAL_RADIO_SetEncryptionAttributes (uint8_t StateMachineNo, uint8_t * enc_iv, uint8_t * enc_key)

Function description

This routines sets the 8-byte encryption initial vector, and the 16-byte encryption key.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **enc_iv:** 8-byte encryption initial vector.
- **enc_key:** 16-byte encryption key.

Return values

- **None:**

HAL_RADIO_SetEncryptFlags

Function name

void HAL_RADIO_SetEncryptFlags (uint8_t StateMachineNo, FunctionalState EncryptFlagTx, FunctionalState EncryptFlagRcv)

Function description

This routine turns encrypt ON and OFF.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **EncryptFlagTx:** DISABLE: encryption is turned OFF for both TX and RX operations. ENABLE: encryption is turned OFF for both TX and RX operations. This parameter can be: ENABLE or DISABLE.
- **EncryptFlagRcv:** encryption is turned OFF for both TX and RX operations. ENABLE: encryption is turned OFF for both TX and RX operations. This parameter can be: ENABLE or DISABLE.

Return values

- **None:**

HAL_RADIO_EncryptPlainData

Function name

void HAL_RADIO_EncryptPlainData (uint8_t * Key, uint8_t * plainData, uint8_t * cypherData)

Function description

Encrypts plain text data using AES encryption with given input key and 128 bit block size.

Parameters

- **Key:** encryption key to be used.
- **plainData:** text data to be encrypted.
- **cypherData:** contains the encrypted data.

Return values

- **None:**

HAL_RADIO_SetDefaultPreambleLen

Function name

void HAL_RADIO_SetDefaultPreambleLen (uint8_t StateMachineNo)

Function description

Restore default preamble length to one byte.

Parameters

- **StateMachineNo:** state machine number in multi state.

Return values

- **None:**

HAL_RADIO_SetPreambleRep

Function name

void HAL_RADIO_SetPreambleRep (uint8_t StateMachineNo, uint8_t PreaLen)

Function description

Set how many times repeat the preamble.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **PreaLen:** preamble length in byte for coded or uncoded phy.

Return values

- **None:**

HAL_RADIO_DisableCRC

Function name

void HAL_RADIO_DisableCRC (uint8_t StateMachineNo, FunctionalState hwCRC)

Function description

Enable or disable the CRC hardware functionality.

Parameters

- **StateMachineNo:** state machine number in multi state.
- **hwCRC:**
  - ENABLE: enable the CRC hardware feature.
  - DISABLE: disable the CRC hardware feature.

Return values

- **None:**

HAL_RADIO_SetNetworkID

Function name

uint8_t HAL_RADIO_SetNetworkID (uint32_t ID)

Function description

This routine sets the network ID field for packet transmission and filtering for the receiving.

Parameters

- **ID:** network ID based on bluetooth specification: 1. It shall have no more than six consecutive zeros or ones. 2. It shall not have all four octets equal. 3. It shall have no more than 24 transitions. 4. It shall have a minimum of two transitions in the most significant six bits.

Return values

- **uint8_t:** return value
  - 0x00 : Success.
  - 0xC0 : Invalid parameter.

HAL_RADIO_CarrierSense

Function name

uint8_t HAL_RADIO_CarrierSense (uint8_t channel, int8_t * rssi)

Function description

HAL_RADIO_Callback

Function name

uint8_t HAL_RADIO_Callback (ActionPacket * p, ActionPacket * next)

Function description

This routine is called when a RADIO event is complete.

Parameters

- **p:** Current action packet which its transaction has been completed.
- **next:** Next action packet which is going to be scheduled.

Return values

- **return:** value: TRUE

HAL_RADIO_CallbackRcvError

Function name

void HAL_RADIO_CallbackRcvError (RxStats_t * rxPacketStats)

Function description

HAL_RADIO_CallbackRcvTimeout

Function name

void HAL_RADIO_CallbackRcvTimeout (RxStats_t * rxPacketStats)

Function description

HAL_RADIO_CallbackRcvEncryptErr

Function name

void HAL_RADIO_CallbackRcvEncryptErr (RxStats_t * rxPacketStats)

Function description

HAL_RADIO_CallbackRcvOk

Function name

void HAL_RADIO_CallbackRcvOk (RxStats_t * rxPacketStats)

Function description

HAL_RADIO_CallbackTxDone

Function name

void HAL_RADIO_CallbackTxDone (void)

Function description

HAL_RADIO_SendPacket

Function name

uint8_t HAL_RADIO_SendPacket (uint8_t channel, uint32_t wakeup_time, uint8_t * txBuffer, uint8_t(*)(ActionPacket *, ActionPacket *) Callback)

Function description

This routine sends a packet on a specific channel and at a specific time.

Parameters

- **channel:** BLE channel index between 0 to 39.
- **wakeup_time:** Time of transmission in us. This is relative time regarding now. Minimum wakeup_time of 230 us. TBR
- **txBuffer:** Pointer to TX data buffer. Second byte of this buffer must be the length of the data.
- **Callback:** This function is being called as data routine. First ActionPacket is current action packet and the second one is next action packet.

Return values

- **uint8_t:** return value
  - 0x00 : Success.
  - 0xC0 : Invalid parameter.
  - 0xC4 : Radio is busy, receiving has not been triggered.

HAL_RADIO_SendPacketWithAck

Function name

uint8_t HAL_RADIO_SendPacketWithAck (uint8_t channel, uint32_t wakeup_time, uint8_t * txBuffer, uint8_t * rxBuffer, uint32_t receive_timeout, uint8_t receive_length, uint8_t(*)(ActionPacket *, ActionPacket *) Callback)

Function description

This routine sends a packet on a specific channel and at a certain time then wait for receiving acknowledge.

Parameters

- **channel:** BLE channel index between 0 to 39.
- **wakeup_time:** Time of transmission based on us. This is relative time regarding now. Minimum wakeup_time of 250 us. TBR
- **txBuffer:** Pointer to TX data buffer. Second byte of this buffer must be the length of the data.
- **rxBuffer:** Pointer to RX data buffer. Second byte of this buffer must be the length of the data.
- **receive_timeout:** Time of RX window used to wait for the packet on us.
- **receive_length:** number of bytes that the link layer accepts in reception.
- **Callback:** This function is being called as data routine. First ActionPacket is current action packet and the second one is next action packet.

Return values

- **uint8_t:** return value
  - 0x00 : Success.
  - 0xC0 : Invalid parameter.
  - 0xC4 : Radio is busy, receiving has not been triggered.

HAL_RADIO_ReceivePacket

Function name

uint8_t HAL_RADIO_ReceivePacket (uint8_t channel, uint32_t wakeup_time, uint8_t * rxBuffer, uint32_t receive_timeout, uint8_t receive_length, uint8_t(*)(ActionPacket *, ActionPacket *) Callback)

Function description

This routine receives a packet on a specific channel and at a certain time.

Parameters

- **channel:** BLE channel index between 0 to 39.
- **wakeup_time:** Time of transmission based on us. This is relative time regarding now. Minimum wakeup_time of 230 us. TBR
- **rxBuffer:** Pointer to RX data buffer. Second byte of this buffer must be the length of the data.
- **receive_timeout:** Time of RX window used to wait for the packet on us.
- **receive_length:** number of bytes that the link layer accepts in reception.
- **Callback:** This function is being called as data routine. First ActionPacket is current action packet and the second one is next action packet.

Return values

- **uint8_t:** return value
  - 0x00 : Success.
  - 0xC0 : Invalid parameter.
  - 0xC4 : Radio is busy, receiving has not been triggered.

HAL_RADIO_ReceivePacketWithAck

Function name

uint8_t HAL_RADIO_ReceivePacketWithAck (uint8_t channel, uint32_t wakeup_time, uint8_t * rxBuffer, uint8_t * txBuffer, uint32_t receive_timeout, uint8_t receive_length, uint8_t(*)(ActionPacket *, ActionPacket *) Callback)

Function description

This routine receives a packet on a specific channel and at a certain time.

Parameters

- **channel:** BLE channel index between 0 to 39.
- **wakeup_time:** time of transmission based on us. This is relative time regarding now. Minimum wakeup_time of 250 us. TBR
- **rxBuffer:** points to received data buffer. second byte of this buffer determines the length of the data.
- **txBuffer:** points to data buffer to send. second byte of this buffer must be the length of the buffer.
- **receive_timeout:** Time of RX window used to wait for the packet on us.
- **receive_length:** number of bytes that the link layer accepts in reception.
- **Callback:** This function is being called as data routine. First ActionPacket is current action packet and the second one is next action packet.

Return values

- **uint8_t:** return value
  - 0x00 : Success.
  - 0xC0 : Invalid parameter.
  - 0xC4 : Radio is busy, receiving has not been triggered.

26 HAL RADIO__TIMER Generic Driver

26.1 RADIO__TIMER Firmware driver registers structures

26.1.1 RADIO_TIMER_InitTypeDef

RADIO_TIMER_InitTypeDef is defined in the stm32wb0x_hal_radio_timer.h

Data Fields

- ***uint16_t XTAL_StartupTime***
- ***bool enableInitialCalibration***
- ***uint32_t periodicCalibrationInterval***

Field Documentation

- ***uint16_t RADIO_TIMER_InitTypeDef::XTAL_StartupTime***
  XTAL startup in 2.44 us unit
- ***bool RADIO_TIMER_InitTypeDef::enableInitialCalibration***
  Enable initial estimation of the frequency of the Low Speed Oscillator, otherwise it will be assumed fixed at 32.768 kHz. Ignored if periodic calibration is active (PeriodicCalibrationInterval != 0).
- ***uint32_t RADIO_TIMER_InitTypeDef::periodicCalibrationInterval***
  Periodic calibration interval in ms, to disable set to 0

26.1.2 VTIMER_HandleTypeS

VTIMER_HandleTypeS is defined in the stm32wb0x_hal_radio_timer.h

Data Fields

- ***uint64_t expiryTime***
- ***VTIMER_CallbackType callback***
- ***bool active***
- ***struct VTIMER_HandleTypeS * next***
- ***void * userData***

Field Documentation

- ***uint64_t VTIMER_HandleTypeS::expiryTime***
  Managed internally when the timer is started
- ***VTIMER_CallbackType VTIMER_HandleTypeS::callback***
  Pointer to the user callback
- ***bool VTIMER_HandleTypeS::active***
  Managed internally when the timer is started
- ***struct VTIMER_HandleTypeS* VTIMER_HandleTypeS::next***
  Managed internally when the timer is started
- ***void* VTIMER_HandleTypeS::userData***
  Pointer to user data

26.2 RADIO__TIMER Firmware driver API description

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

26.2.1 Detailed description of functions

HAL_RADIO_TIMER_Init

Function name

void HAL_RADIO_TIMER_Init (RADIO_TIMER_InitTypeDef * RADIO_TIMER_InitStruct)

Function description

Initialize the radio timer module.

Parameters

- **RADIO_TIMER_InitStruct:** Radio Timer Initialization parameters

Return values

- **None:**

HAL_RADIO_TIMER_Tick

Function name

void HAL_RADIO_TIMER_Tick (void)

Function description

Timer module state machine.

Return values

- **None:**

HAL_RADIO_TIMER_GetRadioTimerStatus

Function name

RADIO_TIMER_Status HAL_RADIO_TIMER_GetRadioTimerStatus (uint64_t * time)

Function description

Return the status of the radio timer.

Parameters

- **time:** Pointer of the variable where the time of the last radio activity scheduled is stored The time is expressed in STU.

Return values

- **0:** if no radio timer is pending.
- **1:** if a radio timer is pending.

HAL_RADIO_TIMER_SetRadioTimerValue

Function name

uint32_t HAL_RADIO_TIMER_SetRadioTimerValue (uint32_t time, uint8_t event_type, uint8_t cal_req)

Function description

Schedules a radio activity for the given absolute timeout value expressed in STU.

Parameters

- **time:** Absolute time expressed in STU.
- **event_type:** Specify if it is a TX (1) or RX (0) event.
- **cal_req:** Specify if PLL calibration is requested (1) or not (0).

Return values

- **0:** if radio activity has been scheduled successfully.
- **1:** if radio activity has been rejected (it is too close or in the past).

HAL_RADIO_TIMER_ClearRadioTimerValue

Function name

uint32_t HAL_RADIO_TIMER_ClearRadioTimerValue (void)

Function description

Clear the last radio activity scheduled disabling the radio timers too.

Return values

- **0:** if the radio activity has been cleared successfully.
- **1:** if it is too late to clear the last radio activity.
- **2:** if it might not be possible to clear the last radio activity.

HAL_RADIO_TIMER_SetRadioTimerRelativeUsValue

Function name

uint32_t HAL_RADIO_TIMER_SetRadioTimerRelativeUsValue (uint32_t rel_timeout_us, bool event_type, bool cal_req)

Function description

Programs Timer1 with a relative timeout - expressed in us - wrt the previous radio event.

Parameters

- **rel_timeout_us:** relative delay, in us, wrt the previous radio event.
- **event_type:** 1 Tx event. 0 Rx event
- **cal_req:** 1 PLL calibartion is requested. 0 PLL calibartion is not requested.

Return values

- **0:** if a correct timeout has been programmed in the timeout register
- **1:** if a correct timeout cannot be programmed

HAL_RADIO_TIMER_GetRadioTimerValue

Function name

uint8_t HAL_RADIO_TIMER_GetRadioTimerValue (uint32_t * time)

Function description

Return the status of the Radio timers and the last value programmed in the register.

Parameters

- **time:** pointer to value which is going to have time value.

Return values

- **0:** if no timer has been programmed.
- **1:** if Timer1 has been programmed.
- **2:** if Timer2 has been programmed.
- **3:** if Wakeup Timer has been programmed.

Notes

- When Timer2 is on schedule, the time is expressed in microseconds, otherwise in absolute machine time units.

HAL_RADIO_TIMER_SetRadioCloseTimeout

Function name

void HAL_RADIO_TIMER_SetRadioCloseTimeout (void)

Function description

Program the radio timer (a.k.a Timer1) as close as possible.

HAL_RADIO_TIMER_RadioTimerIsr

Function name

void HAL_RADIO_TIMER_RadioTimerIsr (void)

Function description

Radio activity finished.

Return values

- **None:**

HAL_RADIO_TIMER_EndOfRadioActivityIsr

Function name

void HAL_RADIO_TIMER_EndOfRadioActivityIsr (void)

Function description

Timer State machine semaphore to signal the radio activity finished.

Return values

- **None:**

HAL_RADIO_TIMER_GetAnchorPoint

Function name

uint64_t HAL_RADIO_TIMER_GetAnchorPoint (uint64_t * current_system_time)

Function description

Get the last anchorPoint in system time unit.

Parameters

- **current_system_time:** Current System Time

HAL_RADIO_TIMER_PowerSaveLevelCheck

Function name

PowerSaveLevels HAL_RADIO_TIMER_PowerSaveLevelCheck (void)

Function description

Returns the admitted low power mode according to the next timer activity.

HAL_RADIO_TIMER_UsToSystime

Function name

uint32_t HAL_RADIO_TIMER_UsToSystime (uint32_t time)

Function description

Translate time in microseconds into sys time units.

Parameters

- **time:** Microseconds to be converted in STU

HAL_RADIO_TIMER_MachineTimeToSysTime

Function name

uint32_t HAL_RADIO_TIMER_MachineTimeToSysTime (uint32_t time)

Function description

Return the STU corresponding to the MTU passed as parameter.

Parameters

- **time:** MTU amount to be converted in STU
- **time:** MTU amount to be converted in STU

HAL_RADIO_TIMER_GetCurrentSysTime

Function name

uint64_t HAL_RADIO_TIMER_GetCurrentSysTime (void)

Function description

This function returns the current reference time expressed in system time units.

HAL_RADIO_TIMER_AddSysTimeMs

Function name

uint64_t HAL_RADIO_TIMER_AddSysTimeMs (uint64_t sysTime, int32_t msTime)

Function description

This function returns the sum of an absolute time and a signed relative time.

Parameters

- **sysTime:** Absolute time expressed in internal time units.
- **msTime:** Signed relative time expressed in ms.

HAL_RADIO_TIMER_DiffSysTimeMs

Function name

int64_t HAL_RADIO_TIMER_DiffSysTimeMs (uint64_t sysTime1, uint64_t sysTime2)

Function description

Returns the difference between two absolute times: sysTime1-sysTime2.

Parameters

- **sysTime2:** Absolute time expressed in internal time units.
- **sysTime1:** Absolute time expressed in internal time units.

HAL_RADIO_TIMER_GetSysTime64

Function name

uint64_t HAL_RADIO_TIMER_GetSysTime64 (uint32_t sys_time)

Function description

Returns the 64-bit system time, referred to the 32-bit system time parameter.

Parameters

- **sys_time:** system time

HAL_RADIO_TIMER_GetFutureSysTime64

Function name

uint64_t HAL_RADIO_TIMER_GetFutureSysTime64 (uint32_t sys_time)

Function description

Returns the next 64-bit system time in the future, referred to the 32-bit system time parameter.

Parameters

- **sys_time:** system time in the future (no more than 10485 s = 174 min in the future)

HAL_RADIO_TIMER_StartVirtualTimer

Function name

uint32_t HAL_RADIO_TIMER_StartVirtualTimer (VTIMER_HandleType * timerHandle, uint32_t msRelTimeout)

Function description

Starts a one-shot virtual timer for the given relative timeout value expressed in ms.

Parameters

- **timerHandle:** The virtual timer
- **msRelTimeout:** The relative time, from current time, expressed in ms

Return values

- **0:** if the timerHandle is valid.
- **1:** if the timerHandle is not valid. It is already started.

HAL_RADIO_TIMER_StartVirtualTimerSysTime

Function name

uint32_t HAL_RADIO_TIMER_StartVirtualTimerSysTime (VTIMER_HandleType * timerHandle, uint64_t time)

Function description

Starts a one-shot virtual timer for the given absolute timeout value expressed in internal system time units.

Parameters

- **timerHandle:** The virtual timer
- **time:** Absolute time expressed in STU.

Return values

- **0:** if the timerHandle is valid.
- **1:** if the timerHandle is not valid. It is already started.

HAL_RADIO_TIMER_StopVirtualTimer

Function name

void HAL_RADIO_TIMER_StopVirtualTimer (VTIMER_HandleType * timerHandle)

Function description

Stops the one-shot virtual timer specified if found.

Parameters

- **timerHandle:** The virtual timer

Return values

- **None:**

HAL_RADIO_TIMER_ExpiryTime

Function name

uint64_t HAL_RADIO_TIMER_ExpiryTime (VTIMER_HandleType * timerHandle)

Function description

Returns the absolute expiry time of a running virtual timer expressed in internal system time units.

Parameters

- **timerHandle:** The virtual timer

Return values

- **sysTime:** Absolute time expressed in internal system time units.

HAL_RADIO_TIMER_TimeoutCallback

Function name

void HAL_RADIO_TIMER_TimeoutCallback (void)

Function description

Virtual timer Timeout Callback.

Return values

- **None:**

HAL_RADIO_TIMER_GetPendingTimers

Function name

uint32_t HAL_RADIO_TIMER_GetPendingTimers (void)

Function description

Returns the number of timers in the queue.

HAL_RADIO_TIMER_CPU_WKUP_IRQHandler

Function name

void HAL_RADIO_TIMER_CPU_WKUP_IRQHandler (void)

Function description

HAL_RADIO_TIMER_TXRX_WKUP_IRQHandler

Function name

void HAL_RADIO_TIMER_TXRX_WKUP_IRQHandler (void)

Function description

HAL_RADIO_TIMER_ERROR_IRQHandler

Function name

void HAL_RADIO_TIMER_ERROR_IRQHandler (void)

Function description

HAL_RADIO_TIMER_CpuWakeUpCallback

Function name

void HAL_RADIO_TIMER_CpuWakeUpCallback (void)

Function description

HAL_RADIO_TIMER_TxRxWakeUpCallback

Function name

void HAL_RADIO_TIMER_TxRxWakeUpCallback (void)

Function description

26.3 RADIO__TIMER Firmware driver defines

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

26.3.1 RADIO__TIMER

RADIO__TIMER

Radio Timer Exported Constants

HAL_RADIO_TIMER_SUCCESS

HAL_RADIO_TIMER_LATE

HAL_RADIO_TIMER_CRITICAL

27 HAL RCC Generic Driver

27.1 RCC Firmware driver registers structures

27.1.1 RCC_OscInitTypeDef

RCC_OscInitTypeDef is defined in the stm32wb0x_hal_rcc.h

Data Fields

- ***uint32_t OscillatorType***
- ***uint32_t HSEState***
- ***uint32_t HSIState***
- ***uint32_t LSEState***
- ***uint32_t LSIState***
- ***uint32_t LSEBYPASSState***

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::HSIState***
  The new state of the HSI. This parameter can be a value of ***RCC_HSI_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::LSIState***
  The new state of the LSI. This parameter can be a value of ***RCC_LSI_Config***
- ***uint32_t RCC_OscInitTypeDef::LSEBYPASSState***
  The new state of the LSE Bypass. This parameter can be a value of ***RCC_LSEBYPASS_Config***

27.1.2 RCC_ClkInitTypeDef

RCC_ClkInitTypeDef is defined in the stm32wb0x_hal_rcc.h

Data Fields

- ***uint32_t SYSCLKSource***
- ***uint32_t SYSCLKDivider***

Field Documentation

- ***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 clock divider used as system clock (SYSCLK). This parameter can be a value of ***RCC_RC64MPLL_Clock_Divider*** or ***RCC_DIRECT_HSE_Clock_Divider***

27.2 RCC Firmware driver API description

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

27.2.1 RCC specific features

After reset the device is running from High Speed Internal oscillator (64 MHz RC oscillator) with flash 1 wait state. All peripherals clock are off except internal SRAM, flash and GPIO for SWD communication.

- The prescaler is DIV 4
- The clock for all peripherals is switched off, except the SRAM and FLASH and GPIO.
- All GPIOs are in input mode, except the SWD pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and flash settings
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals which clocks are not always 16MHz or 32 MHz (SMPS, SPI2I2S, SPI3I2S)

27.2.2 Initialization and de-initialization functions

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

Internal/external clock and PLL configuration

- HSI (high-speed internal): 64 MHz factory-trimmed RC used through the PLL as System clock source.
- LSI (low-speed internal): 32 KHz low consumption RC used as IWDG, LPUART, RF system Auto-wakeup from DeepStop modes. clock source.
- HSE (high-speed external): 32 MHz crystal oscillator.
- LSE (low-speed external): 32.768 KHz oscillator used as IWDG, RTC, LPUART, RF system Auto-wakeup from DeepStop modes.
- PLL (clocked by HSI, HSE) providing system clock.
- MCO (microcontroller clock output): used to output HSI, HSE, SYSCLK, HSI64M_DIV2048, RC64MPLL clock (through a configurable prescaler) on PA5, PA11 & PB15 pins.

Note: All the peripheral clocks have an always 16 MHz or 32 MHz to maintain fixed baud rate while system clock is switching from a frequency to another from the System clock (SYSCLK). An always 32 or 16 MHz requested by few peripherals like the MR_BLE radio IP for instance. An always 16 MHz requested by few peripherals like serial interfaces or like flash controller and MR_BLE radio IP (to have a fixed reference clock to manage delays).

  - *A programmable prescaled clock for I2S block, that can be 16 MHz / 32 MHz. You have to use @ref __HAL_RCC_SPI2I2S_CONFIG(), @ref __HAL_RCC_SPI3I2S_CONFIG and @ref HAL_RCCEx_PeriphCLKConfig() function to configure this clock.*

- The maximum frequency of the SYSCLK is 64 MHz. The maximum frequency of the CLK_SYS_BLE is 32 MHz.

This section contains the following APIs:

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

27.2.3 Peripheral Control functions

This subsection provides a set of functions allowing to:

- Output clock to MCO pin.
- Retrieve current clock frequencies.

This section contains the following APIs:

- ***HAL_RCC_MCOConfig()***
- ***HAL_RCC_GetSysClockFreq()***
- ***HAL_RCC_GetOscConfig()***
- ***HAL_RCC_GetClockConfig()***

27.2.4 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 16MHz, PLL OFF
- This function doesn't modify the configuration of the Peripheral clocksLSI and LSE clocks

HAL_RCC_OscConfig

Function name

HAL_StatusTypeDef HAL_RCC_OscConfig (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

HAL_RCC_ClockConfig

Function name

HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)

Function description

Initialize the system clock 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_WAIT_STATES_0 FLASH 0 wait state cycle
  – FLASH_WAIT_STATES_1 FLASH 1 wait state cycle

Return values

- **HAL:** status

Notes

- The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function
- 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.

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 MCO1 pin(PA5) or MC02 pin (PA11) or MCO3 pin (PB14/PB15).

Parameters

- **RCC_MCOx:** specifies the output direction for the clock source.
  - RCC_MCO1 Clock source to output on MCO1 pin(PA5)
  - RCC_MCO2 Clock source to output on MCO2 pin(PA11)
  - RCC_MCO3 Clock source to output on MCO3 pin(PB14/PB15)
- **RCC_MCOSource:** specifies the clock source to output. This parameter can be one of the following values:
  - RCC_MCOSOURCE_NOCLOCK MCO output disabled, no clock on MCO
  - 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_MCOSOURCE_RC64MPLL RC64MPLL clock selected as MCO source
  - RCC_MCOSOURCE_HSI64M_DIV2048 HSI64M_DIV2048 clock selected as MCO source
  - RCC_MCOSOURCE_SMPS SMPS clock selected as MCO source
  - RCC_MCOSOURCE_ADC ADC clock before stabilization selected as MCO 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

Return values

- **None:**

Notes

- PA5, PA11 or PB14/PB15 should be configured in alternate function mode.

HAL_RCC_GetSysClockFreq

Function name

uint32_t HAL_RCC_GetSysClockFreq (void)

Function description

Return the SYSCLK frequency.

Return values

- **SYSCLK:** frequency

Notes

- The system computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source. The return value is the content of the SystemCoreClock CMSIS variable

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:**

27.3 RCC Firmware driver defines

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

27.3.1 RCC

RCC

AHB Peripheral Force Release Reset

__HAL_RCC_AHB1_FORCE_RESET

__HAL_RCC_DMA_FORCE_RESET

__HAL_RCC_GPIOA_FORCE_RESET

__HAL_RCC_GPIOB_FORCE_RESET

__HAL_RCC_CRC_FORCE_RESET

__HAL_RCC_PKA_FORCE_RESET

__HAL_RCC_RNG_FORCE_RESET

__HAL_RCC_AHB1_RELEASE_RESET

__HAL_RCC_DMA_RELEASE_RESET

__HAL_RCC_GPIOA_RELEASE_RESET

__HAL_RCC_GPIOB_RELEASE_RESET

__HAL_RCC_CRC_RELEASE_RESET

__HAL_RCC_PKA_RELEASE_RESET

__HAL_RCC_RNG_RELEASE_RESET

AHB Peripheral Clock Enable Disable

__HAL_RCC_DMA_CLK_ENABLE

__HAL_RCC_GPIOA_CLK_ENABLE

__HAL_RCC_GPIOB_CLK_ENABLE

__HAL_RCC_CRC_CLK_ENABLE

__HAL_RCC_PKA_CLK_ENABLE

__HAL_RCC_RNG_CLK_ENABLE

__HAL_RCC_DMA_CLK_DISABLE

__HAL_RCC_GPIOA_CLK_DISABLE

__HAL_RCC_GPIOB_CLK_DISABLE

__HAL_RCC_CRC_CLK_DISABLE

__HAL_RCC_PKA_CLK_DISABLE

__HAL_RCC_RNG_CLK_DISABLE

AHB Peripheral Clock Enabled or Disabled Status

__HAL_RCC_DMA_IS_CLK_ENABLED

__HAL_RCC_GPIOA_IS_CLK_ENABLED

__HAL_RCC_GPIOB_IS_CLK_ENABLED

__HAL_RCC_CRC_IS_CLK_ENABLED

__HAL_RCC_PKA_IS_CLK_ENABLED

__HAL_RCC_RNG_IS_CLK_ENABLED

__HAL_RCC_DMA_IS_CLK_DISABLED

__HAL_RCC_GPIOA_IS_CLK_DISABLED

__HAL_RCC_GPIOB_IS_CLK_DISABLED

__HAL_RCC_CRC_IS_CLK_DISABLED

__HAL_RCC_PKA_IS_CLK_DISABLED

__HAL_RCC_RNG_IS_CLK_DISABLED

APB0 Peripheral Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE

__HAL_RCC_SYSCFG_CLK_ENABLE

__HAL_RCC_RTC_CLK_ENABLE

__HAL_RCC_WDG_CLK_ENABLE

__HAL_RCC_TIM16_CLK_ENABLE

__HAL_RCC_TIM17_CLK_ENABLE

__HAL_RCC_TIM2_CLK_DISABLE

__HAL_RCC_SYSCFG_CLK_DISABLE

__HAL_RCC_RTC_CLK_DISABLE

__HAL_RCC_WDG_CLK_DISABLE

__HAL_RCC_TIM16_CLK_DISABLE

__HAL_RCC_TIM17_CLK_DISABLE

APB0 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_TIM2_IS_CLK_ENABLED

__HAL_RCC_SYSCFG_IS_CLK_ENABLED

__HAL_RCC_RTC_IS_CLK_ENABLED

__HAL_RCC_WDG_IS_CLK_ENABLED

__HAL_RCC_TIM16_IS_CLK_ENABLED

__HAL_RCC_TIM17_IS_CLK_ENABLED

__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_SYSCFG_IS_CLK_DISABLED

__HAL_RCC_RTC_IS_CLK_DISABLED

__HAL_RCC_WDG_IS_CLK_DISABLED

__HAL_RCC_TIM16_IS_CLK_DISABLED

__HAL_RCC_TIM17_IS_CLK_DISABLED

APB0 Peripheral Force Release Reset

__HAL_RCC_APB0_FORCE_RESET

__HAL_RCC_TIM2_FORCE_RESET

__HAL_RCC_SYSCFG_FORCE_RESET

__HAL_RCC_RTC_FORCE_RESET

__HAL_RCC_WDG_FORCE_RESET

__HAL_RCC_TIM16_FORCE_RESET

__HAL_RCC_TIM17_FORCE_RESET

__HAL_RCC_APB0_RELEASE_RESET

__HAL_RCC_TIM2_RELEASE_RESET

__HAL_RCC_SYSCFG_RELEASE_RESET

__HAL_RCC_RTC_RELEASE_RESET

__HAL_RCC_WDG_RELEASE_RESET

__HAL_RCC_TIM16_RELEASE_RESET

__HAL_RCC_TIM17_RELEASE_RESET

APB1 Peripheral Clock Enable Disable

__HAL_RCC_ADCDIG_CLK_ENABLE

__HAL_RCC_ADCANA_CLK_ENABLE

__HAL_RCC_LPUART1_CLK_ENABLE

__HAL_RCC_USART1_CLK_ENABLE

__HAL_RCC_SPI3_CLK_ENABLE

__HAL_RCC_I2C1_CLK_ENABLE

__HAL_RCC_ADCDIG_CLK_DISABLE

__HAL_RCC_ADCANA_CLK_DISABLE

__HAL_RCC_LPUART1_CLK_DISABLE

__HAL_RCC_USART1_CLK_DISABLE

__HAL_RCC_SPI3_CLK_DISABLE

__HAL_RCC_I2C1_CLK_DISABLE

APB1 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_ADCDIG_IS_CLK_ENABLED

__HAL_RCC_ADCANA_IS_CLK_ENABLED

__HAL_RCC_LPUART1_IS_CLK_ENABLED

__HAL_RCC_USART1_IS_CLK_ENABLED

__HAL_RCC_SPI3_IS_CLK_ENABLED

__HAL_RCC_I2C1_IS_CLK_ENABLED

__HAL_RCC_ADCDIG_IS_CLK_DISABLED

__HAL_RCC_ADCANA_IS_CLK_DISABLED

__HAL_RCC_LPUART1_IS_CLK_DISABLED

__HAL_RCC_USART1_IS_CLK_DISABLED

__HAL_RCC_SPI3_IS_CLK_DISABLED

__HAL_RCC_I2C1_IS_CLK_DISABLED

APB1 Peripheral Force Release Reset

__HAL_RCC_APB1_FORCE_RESET

__HAL_RCC_ADC_FORCE_RESET

__HAL_RCC_LPUART1_FORCE_RESET

__HAL_RCC_USART1_FORCE_RESET

__HAL_RCC_SPI3_FORCE_RESET

__HAL_RCC_I2C1_FORCE_RESET

__HAL_RCC_APB1_RELEASE_RESET

__HAL_RCC_ADC_RELEASE_RESET

__HAL_RCC_LPUART1_RELEASE_RESET

__HAL_RCC_USART1_RELEASE_RESET

__HAL_RCC_SPI3_RELEASE_RESET

__HAL_RCC_I2C1_RELEASE_RESET

APB2 Peripheral Clock Enable Disable

__HAL_RCC_RADIO_CLK_ENABLE

__HAL_RCC_RADIO_CLK_DISABLE

APB2 Peripheral Clock Enabled or Disabled Status

__HAL_RCC_RADIO_IS_CLK_ENABLED

__HAL_RCC_RADIO_IS_CLK_DISABLED

APB2 Peripheral Force Release Reset

__HAL_RCC_APB2_FORCE_RESET

__HAL_RCC_RADIO_FORCE_RESET

__HAL_RCC_APB2_RELEASE_RESET

__HAL_RCC_RADIO_RELEASE_RESET

Clock Source Configuration Macros

__HAL_RCC_RC64MPLL_ENABLE

Description:

- Macros to enable or disable the main RC64MPLL.

Return value:

- None

Notes:

- After enabling the main RC64MPLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main RC64MPLL can not be disabled if it is used as system clock source The main RC64MPLL is disabled by hardware when entering DEEPSTOP and SHUTDOWN modes.

__HAL_RCC_RC64MPLL_DISABLE

__HAL_RCC_RC64MPLL_READYFLAG

Description:

- Macro to get the RC64MPLL ready flag.

Return value:

- Returns: the Ready Flag status

__HAL_RCC_RC64MPLL_PRESC_CONFIG

Description:

- Macro to configure the RC64MPLL division factor.

Parameters:

- __PLLD__: specifies the division factor This parameter must be a value of RCC_RC64MPLL_Clock_Divider.

Return value:

- None

Notes:

- This function must be used only when the main RC64MPLL is disabled.

__HAL_RCC_DIRECT_HSE_PRESC_CONFIG

Description:

- Macro to configure the DIRERCT HSE division factor.

Parameters:

- __DIRECTHSED__: specifies the division factor This parameter must be a value of RCC_DIRECT_HSE_Clock_Divider.

Return value:

- None

Notes:

- This function must be used only when the main DIRECT HSE is disabled.

__HAL_RCC_RF_CLK_SET_CONFIG

Description:

- Macro to configure the RF clock frequency.

Parameters:

- __FREQ__: specifies the RF BLE clock frequency. This parameter must be a value of RCC_RF_Clock.

Return value:

- None

__HAL_RCC_RF_CLK_GET_CONFIG

Description:

- Macro to get the RF BLE clock source.

Return value:

- The: RF BLE clock source. The returned value can be one of the following values:
  - RCC_RF_CLK_16M
  - RCC_RF_CLK_32M

__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_RC64MPLL RC64PLL configuration is used as system clock source.
  - RCC_SYSCLKSOURCE_DIRECT_HSE DIRECT HSE configuration is used as system clock source.

Return value:

- None

__HAL_RCC_GET_SYSCLK_SOURCE

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
  - RCC_SYSCLKSOURCE_RC64MPLL RC64PLL configuration is used as system clock source..
  - RCC_SYSCLKSOURCE_DIRECT_HSE DIRECT HSE configuration is used as system clock source.

Clock Switch Divider

RCC_RC64MPLL_SWITCH_DIV1

RC64MPLL division factor = 1

RCC_RC64MPLL_SWITCH_DIV2

RC64MPLL division factor = 2

RCC_RC64MPLL_SWITCH_DIV4

RC64MPLL division factor = 4

RCC_DIRECT_HSE_SWITCH_DIV1

Direct HSE division factor = 1

RCC_DIRECT_HSE_SWITCH_DIV2

Direct HSE division factor = 2

Clock switch macros

__HAL_RCC_CLOCK_SWITCH

Description:

- Set RC64MPLL prescaler to switch the clock when the MR_BLE is enabled.

Parameters:

- __PLLD__: specifies the DIRECT_HSE or RC64MPLL value to setup. This parameter can be one of the following values:
  - RCC_DIRECT_HSE_SWITCH_DIV1 DIRECT HSE division factor = 1
  - RCC_DIRECT_HSE_SWITCH_DIV2 DIRECT HSE division factor = 2
  - RCC_RC64MPLL_SWITCH_DIV1 RC64MPLL division factor = 1
  - RCC_RC64MPLL_SWITCH_DIV2 RC64MPLL division factor = 2
  - RCC_RC64MPLL_SWITCH_DIV4 RC64MPLL division factor = 4

Return value:

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

Notes:

- The CPU/system frequency must be kept at minimum 16 MHz clock when the radio is used.

__HAL_RCC_CLOCK_SWITCH_IT_ENABLE

Description:

- Macros to enable or disable the interrupt of the CPU / system clock frequency switch.

Return value:

- Status: of the interrupt of the CPU / system clock frequency switch

__HAL_RCC_CLOCK_SWITCH_IT_DISABLE

__HAL_RCC_CLOCK_SWITCH_INTERRUPT_IS_ENABLED

Description:

- Macros to read the configuration status of the interrupt of the CPU / system clock frequency switch.

Return value:

- Interrupt: Enable status

__HAL_RCC_CLOCK_SWITCH_INTERRUPT_STATUS

Description:

- Macros to read the status of the interrupt of the CPU / system clock frequency switch.

Return value:

- Interrupt: status

__HAL_RCC_CLOCK_SWITCH_CLEAR_INTERRUPT_STATUS

Description:

- Macros to clear the status of the interrupt of the CPU / system clock frequency switch.

Return value:

- Clear: Interrupt status

DIRECT HSE Clock Divider

RCC_DIRECT_HSE_DIV1

DIRECT HSE division factor = 1

RCC_DIRECT_HSE_DIV2

DIRECT HSE division factor = 2

RCC_DIRECT_HSE_DIV4
DIRECT HSE division factor = 4

RCC_DIRECT_HSE_DIV8
DIRECT HSE division factor = 8

RCC_DIRECT_HSE_DIV16
DIRECT HSE division factor = 16

RCC_DIRECT_HSE_DIV32
DIRECT HSE division factor = 32

LL_RCC_DIRECT_HSE_DIV_1
DIRECT HSE division factor = 1

LL_RCC_DIRECT_HSE_DIV_2
DIRECT HSE division factor = 2

LL_RCC_DIRECT_HSE_DIV_4
DIRECT HSE division factor = 4

LL_RCC_DIRECT_HSE_DIV_8
DIRECT HSE division factor = 8

LL_RCC_DIRECT_HSE_DIV_16
DIRECT HSE division factor = 16

LL_RCC_DIRECT_HSE_DIV_32
DIRECT HSE division factor = 32

Flags

RCC_FLAG_HSIRDY
HSI Ready flag

RCC_FLAG_HSERDY
HSE Ready flag

RCC_FLAG_PLLRDY
PLL Ready flag

RCC_FLAG_LSIRDY
LSI Ready flag

RCC_FLAG_LSERDY
LSE Ready flag

RCC_FLAG_LOCKUPRST
CPU lockup reset flag

RCC_FLAG_WDGRST
Watchdog reset flag

RCC_FLAG_SFTRST
Software reset flag

RCC_FLAG_PORRST
Power-On or BOR reset flag

RCC_FLAG_PADRSTF

NRSTn pad 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 enable
  - RCC_IT_LSERDY LSE ready interrupt enable
  - RCC_IT_HSIRDY HSI ready interrupt enable
  - RCC_IT_HSERDY HSE ready interrupt enable
  - RCC_IT_PLLRDY Main PLL ready interrupt enable
  - RCC_IT_PLL_UNLOCKRDY PLL unlock interrupt enable
  - RCC_IT_RTCRSTRELRDY RTC Reset Release ready interrupt enable
  - RCC_IT_WDGRSTRELRDY WDG Reset Release ready interrupt enable
  - RCC_IT_LPURSTRELRDY LPU Reset Release ready interrupt enable

Return value:

- None

__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 disable
  - RCC_IT_LSERDY LSE ready interrupt disable
  - RCC_IT_HSIRDY HSI ready interrupt disable
  - RCC_IT_HSERDY HSE ready interrupt disable
  - RCC_IT_PLLRDY Main PLL ready interrupt disable
  - RCC_IT_PLL_UNLOCKRDY PLL Unlock interrupt disable
  - RCC_IT_RTCRSTRELRDY RTC Reset Release ready interrupt disable
  - RCC_IT_WDGRSTRELRDY WDG Reset Release ready interrupt disable
  - RCC_IT_LPURSTRELRDY LPU Reset Release ready interrupt disable

Return value:

- None

__HAL_RCC_CLEAR_IT

Description:

- Clear RCC interrupt pending bits (Perform Byte access to RCC_CICR[17:0] bits to clear the selected interrupt pending bits.

Parameters:

- __INTERRUPT__: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
  - RCC_IT_LSIRDY LSI ready interrupt clear
  - RCC_IT_LSERDY LSE ready interrupt clear
  - RCC_IT_HSIRDY HSI ready interrupt clear
  - RCC_IT_HSERDY HSE ready interrupt clear
  - RCC_IT_PLLRDY Main PLL ready interrupt clear
  - RCC_IT_PLL_UNLOCKRDY PLL Unlock interrupt clear
  - RCC_IT_RTCRSTRELRDY RTC Reset Release ready interrupt clear
  - RCC_IT_WDGRSTRELRDY WDG Reset Release ready interrupt clear
  - RCC_IT_LPURSTRELRDY LPU Reset Release ready interrupt clear

__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 flag
  - RCC_IT_LSERDY LSE ready interrupt flag
  - RCC_IT_HSIRDY HSI ready interrupt flag
  - RCC_IT_HSERDY HSE ready interrupt flag
  - RCC_IT_PLLRDY Main PLL ready interrupt flag
  - RCC_IT_PLL_UNLOCKRDY PLL Unlock interrupt flag
  - RCC_IT_RTCRSTRELRDY RTC Reset Release ready interrupt flag
  - RCC_IT_WDGRSTRELRDY WDG Reset Release ready interrupt flag
  - RCC_IT_LPURSTRELRDY LPU Reset Release ready interrupt flag

Return value:

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

HSE bias current factor

RCC_HSEAMPTHRESHOLD_1_2

HSE bias current factor 1/2

RCC_HSEAMPTHRESHOLD_3_4

HSE bias current factor 3/4

External high speed clock buffer for PLL RF2G4

__HAL_RCC_HSEPLLBUFON_ENABLE

Description:

- Macros to enable or disable the high speed clock buffer for PLL RF2G4.

Return value:

- Status: of the high speed clock buffer for PLL RF2G4

__HAL_RCC_HSEPLLBUFON_DISABLE

__HAL_RCC_HSEPLLBUFON_IS_ENABLED

Description:

- Macros to read the status of the high speed clock buffer for PLL RF2G4.

Return value:

- Configuration: status

HSE Config

RCC_HSE_OFF

HSE clock deactivation

RCC_HSE_ON

HSE clock activation

HSE current max limit

RCC_HSE_CURRENTMAX_0

HSE current max limit 0.18 mA/V

RCC_HSE_CURRENTMAX_1

HSE current max limit 0.57 mA/V

RCC_HSE_CURRENTMAX_2

HSE current max limit 0.78 mA/V

RCC_HSE_CURRENTMAX_3

HSE current max limit 1.13 mA/V

RCC_HSE_CURRENTMAX_4

HSE current max limit 0.61 mA/V

RCC_HSE_CURRENTMAX_5

HSE current max limit 1.65 mA/V

RCC_HSE_CURRENTMAX_6

HSE current max limit 2.12 mA/V

RCC_HSE_CURRENTMAX_7

HSE current max limit 2.84 mA/V

HSE Macros

__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.
  - RCC_HSE_ON Turn ON the HSE oscillator.

Return value:

- None

Notes:

- After enabling the HSE (RCC_HSE_ON), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the RC64MPLL. The HSE is stopped by hardware when entering DEEPSTOP and SHUTDOWN modes.

__HAL_RCC_GET_HSE_READYFLAG

Description:
- Macro to get the HSE ready flag.

Return value:
- Returns: the Ready Flag status

__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.
  - RCC_LSE_ON Turn ON the LSE oscillator.

Return value:
- None

Notes:
- After enabling the LSE (RCC_LSE_ON), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used.

__HAL_RCC_GET_LSE_READYFLAG

Description:
- Macro to get the LSE ready flag.

Return value:
- Returns: the Ready Flag status

__HAL_RCC_LSE_BYPASS_CONFIG

Description:
- Macro to configure the External Low Speed oscillator (LSE) in Bypass mode.

Parameters:
- __STATE__: specifies the new state of the LSE bypass configuration. This parameter can be one of the following values:
  - RCC_LSE_BYPASS_OFF Turn OFF the LSE bypass configuration.
  - RCC_LSE_BYPASS_ON Turn ON the LSE bypass configuration.

Return value:
- None

__HAL_RCC_HSE_AMPCONFIG

Description:
- Macros to configure HSE sense amplifier threshold.

Parameters:
- __HSE_AMPTHRES__: specifies the HSE sense amplifier threshold. This parameter can be one of the following values:
  - RCC_HSEAMPTHRESHOLD_1_2 HSE bias current factor 1/2.
  - RCC_HSEAMPTHRESHOLD_3_4 HSE bias current factor 3/4.

Return value:
- None

Notes:
- to configure HSE sense amplifier, first disable HSE using __HAL_RCC_HSE_CONFIG(RCC_HSE_OFF) macro.

__HAL_RCC_HSE_CURRENTCONFIG

Description:
- Macros to configure HSE current control.

Parameters:
- __HSE_CURRENTMAX__: specifies the HSE current max limit. This parameter can be one of the following values:
  - RCC_HSE_CURRENTMAX_0 HSE current max limit 0.18 mA/V.
  - RCC_HSE_CURRENTMAX_1 HSE current max limit 0.57 mA/V.
  - RCC_HSE_CURRENTMAX_2 HSE current max limit 0.78 mA/V.
  - RCC_HSE_CURRENTMAX_3 HSE current max limit 1.13 mA/V.
  - RCC_HSE_CURRENTMAX_4 HSE current max limit 0.61 mA/V.
  - RCC_HSE_CURRENTMAX_5 HSE current max limit 1.65 mA/V.
  - RCC_HSE_CURRENTMAX_6 HSE current max limit 2.12 mA/V.
  - RCC_HSE_CURRENTMAX_7 HSE current max limit 2.84 mA/V.

Return value:
- None

Notes:
- to configure HSE current control, first disable HSE using __HAL_RCC_HSE_CONFIG(RCC_HSE_OFF) macro.

__HAL_RCC_HSE_CAPACITORTUNING

Description:
- Macros to configure HSE capacitor tuning.

Parameters:
- __HSE_LOAD_CAPACITANCE__: specifies the HSE capacitor value. This Value Between Min_Data = 0 and Max_Data = 63

Return value:
- None

Notes:
- to configure HSE current control, first disable HSE using __HAL_RCC_HSE_CONFIG(RCC_HSE_OFF) macro.

__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

HSI default calibration trimming value

RCC_HSICALIBRATION_DEFAULT

Default HSI calibration trimming value

HSI Config

RCC_HSI_OFF

HSI clock activation

RCC_HSI_ON

HSI clock deactivation

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_PLLRDY

PLL Ready Interrupt flag

RCC_IT_PLL_UNLOCK

PLL Unlock Interrupt flag

RCC_IT_RTCRSTRELRDY

RTC Reset Release Interrupt Flag

RCC_IT_WDGRSTRELRDY

WDG Reset Release Interrupt Flag

RCC_IT_LPURSTRELRDY

LPU Reset Release Interrupt Flag

RCC Private macros to check input parameters

IS_RCC_HSE

IS_RCC_HSI

IS_RCC_LSE

IS_RCC_LSE_BYPASS

IS_RCC_LSI

IS_RCC_SYSCLKSOURCE

IS_RCC_SYSCLK_DIVIDER

IS_RCC_MCO

IS_RCC_MCO1SOURCE

IS_RCC_MCODIV

IS_RCC_LSE_DRIVE

LSCO Clock Source

RCC_LSCOSOURCE_NOCLOCK

LSCO output disabled, no clock on LSCO

RCC_LSCOSOURCE_LSI

LSI selection as LSCO source

RCC_LSCOSOURCE_LSE

LSE selection as LSCO source

LSCO Index

RCC_LSCO1

LSCO1 index

RCC_LSCO2

LSCO2 index

RCC_LSCO3

LSCO3 index

LSE BYPASS Config

RCC_LSE_BYPASS_OFF

LSE clock deactivation

RCC_LSE_BYPASS_ON

LSE clock activation

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

LSI Config

RCC_LSI_OFF

LSI clock deactivation

RCC_LSI_ON

LSI clock activation

LSI Macros

__HAL_RCC_HSI_CONFIG

Description:

- Macro to configure the Internal High Speed oscillator (HSI).

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.
  - RCC_HSE_ON Turn ON the HSE oscillator.

Return value:

- None

Notes:

- After enabling the HSI (RCC_HSI_ON), the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable. The HSI is stopped by hardware when entering DEEPSTOP and SHUTDOWN modes.

__HAL_RCC_GET_HSI_READYFLAG

Description:

- Macro to get the HSI ready flag.

Return value:

- Returns: the Ready Flag status

__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 Min_data=0 and Max_Data=63.

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_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. LSI cannot be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI1 oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

__HAL_RCC_GET_LSI_READYFLAG

Description:

- Macro to get the LSI ready flag.

Return value:

- Returns: the Ready Flag status

__HAL_RCC_LSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro to adjust the Internal Low Speed oscillator (LSI) calibration value.

Parameters:

- __LSITRIMMINGVALUE__: specifies the calibration trimming value This parameter must be a number between Min_data=0 and Max_Data=15.

Return value:

- None

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

MCO 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

MCO Clock Source

RCC_MCOSOURCE_NOCLOCK

MCO output disabled, no clock on MCO

RCC_MCO1SOURCE_SYSCLK

SYSCLK selection as MCO source

RCC_MCO1SOURCE_HSI

HSI selection as MCO source

RCC_MCO1SOURCE_HSE

HSE after stabilization selection as MCO source

RCC_MCOSOURCE_RC64MPLL

RC64MPLL selection as MCO source

RCC_MCOSOURCE_HSI64M_DIV2048

HSI64M_DIV2048 selection as MCO source

RCC_MCOSOURCE_SMPS

SMPS selection as MCO source

RCC_MCOSOURCE_ADC

ADC selection as MCO source

Main Configurable Clock Output Selection Macros

__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_MCOSOURCE_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_MCOSOURCE_RC64MPLL Main PLL clock selected as MCO source
  – RCC_MCOSOURCE_HSI64M_DIV2048 HSI div 2048 clock selected as MCO source
  – RCC_MCOSOURCE_SMPS SMPS clock selected as MCO source
  – RCC_MCOSOURCE_ADC ADC clock selected as MCO source
- __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

MCO Index

RCC_MCO1

MCO1 index

RCC_MCO2

MCO2 index

RCC_MCO3

MCO3 index

Oscillator Type

RCC_OSCILLATORTYPE_HSI

Oscillator configuration unchanged

RCC_OSCILLATORTYPE_NONE

Oscillator configuration unchanged

RCC_OSCILLATORTYPE_HSE

HSE to configure

RCC_OSCILLATORTYPE_LSE

LSE to configure

RCC_OSCILLATORTYPE_LSI

LSI to configure

RCC_OSCILLATORTYPE_LSE_BYPASS

Slow clock is directly provided

RC64MPLL Clock Divider

RCC_RC64MPLL_DIV1

RC64MPLL division factor = 1

RCC_RC64MPLL_DIV2

RC64MPLL division factor = 2

RCC_RC64MPLL_DIV4

RC64MPLL division factor = 4

RCC_RC64MPLL_DIV8

RC64MPLL division factor = 8

RCC_RC64MPLL_DIV16

RC64MPLL division factor = 16

RCC_RC64MPLL_DIV32

RC64MPLL division factor = 32

RCC_RC64MPLL_DIV64

RC64MPLL division factor = 64

Rest Status Flags Management

__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 Reset flag is set or not.

Parameters:

- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
  – RCC_FLAG_LOCKUPRST CPU lockup reset flag
  – RCC_FLAG_WDGRST Watchdog reset flag
  – RCC_FLAG_SFTRST Software reset flag
  – RCC_FLAG_PORRST Power-On or BOR reset flag
  – RCC_FLAG_PADRSTF NRSTn pad reset flag

Return value:

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

RF Clock

RCC_RF_CLK_16M

RF system clock 16 MHz

RCC_RF_CLK_32M

RF system clock 32 MHz

System Clock Source

RCC_SYSCLKSOURCE_RC64MPLL

RC64MPLL system clock configuration

RCC_SYSCLKSOURCE_DIRECT_HSE

DIRECT HSE system clock configuration

RCC_SYSCLKSOURCE_HSI

RC64MPLL system clock configuration

Timeout Values

RCC_LSE_TIMEOUT_VALUE

28 HAL RCC Extension Driver

28.1 RCCEx Firmware driver registers structures

28.1.1 RCC_PeriphCLKInitTypeDef

RCC_PeriphCLKInitTypeDef is defined in the stm32wb0x_hal_rcc_ex.h

Data Fields

- ***uint32_t PeriphClockSelection***
- ***uint32_t SPI3I2SClockSelection***
- ***uint32_t RTCWDGBLEWKUPClockSelection***
- ***uint32_t LPUART1ClockSelection***
- ***uint32_t RFClockSelection***
- ***uint32_t SmpsDivSelection***

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::SPI3I2SClockSelection***
  Specifies SPI3_I2S clock source. This parameter can be a value of ***RCCEx_SPI3_I2S_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RTCWDGBLEWKUPClockSelection***
  Specifies RTC, WDG and BLEWKUP clock source. This parameter can be a value of ***RCCEx_RTC_WDG_BLEWKUP_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::LPUART1ClockSelection***
  Specifies LPUART1 clock source. This parameter can be a value of ***RCCEx_LPUART1_Clock_Source***
- ***uint32_t RCC_PeriphCLKInitTypeDef::RFClockSelection***
  RADIO IP does not use directly the system clock for its APB / AHB interfaces but the system clock with a potential divider (2 or 4). This parameter can be a value of ***RCC_RF_Clock*** The supported configurations are:
  – CLK_SYS = 1 MHz / 2 MHz / 4 MHz / 8 MHz --> Not possible to use RADIO IP
  – CLK_SYS = 64 MHz / 32 MHz / 16 MHz --> Allow the use of the RADIO IP. The only constraint is that the CLK_SYS shall be bigger or equal than RADIO IP clk
- ***uint32_t RCC_PeriphCLKInitTypeDef::SmpsDivSelection***
  Specifies SMPS clock prescaling factor. This parameter can be a value of ***RCCEx_SMPS_Clock_Divider***

28.2 RCCEx Firmware driver API description

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

28.2.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

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

28.2.2 Extended clock management functions

This subsection provides a set of functions allowing to control the activation or deactivation Low speed clock output.

This section contains the following APIs:

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

- *HAL_RCCEx_TrimOsc()*

28.2.3 Detailed description of functions

HAL_RCCEx_PeriphCLKConfig

Function name

HAL_StatusTypeDef HAL_RCCEx_PeriphCLKConfig (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:

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(RF, SMPS, SPI2I2S, SPI3I2S).

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 with clock source.

Parameters

- **PeriphClk:** Peripheral clock identifier This parameter can be one of the following values:
  - RCC_PERIPHCLK_RF RF peripheral clock
  - RCC_PERIPHCLK_SMPS SMPS peripheral clock
  - RCC_PERIPHCLK_SPI2_I2S SPI2 I2S peripheral clock (*)
  - RCC_PERIPHCLK_SPI3_I2S SPI3 I2S peripheral clock (*)
  - RCC_PERIPHCLK_LPUART1 LPUART1 peripheral clock (*)
  - RCC_PERIPHCLK_RTC_WDG_BLEWKUP RTC, WDG and BLEWKUP peripheral clock

Return values

- **Frequency:** in Hz

Notes

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

HAL_RCCEx_EnableLSCO

Function name

void HAL_RCCEx_EnableLSCO (uint32_t RCC_LSCOx, uint32_t RCC_LSCOSource)

Function description

Select the clock source to output on LSCO1 pin(PA4) or LSC02 pin (PA10) or LSCO3 pin (PB12).

Parameters

- **RCC_LSCOx:** specifies the output direction for the clock source.
  - RCC_LSCO1 Clock source to output on LSCO1 pin(PA4)
  - RCC_LSCO2 Clock source to output on LSCO2 pin(PA10)
  - RCC_LSCO3 Clock source to output on LSCO3 pin(PB12)
- **RCC_LSCOSource:** specifies the 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:**

Notes

- PA4, PA10 or PB12 should be configured in alternate function mode.
- LSCO should be disable with HAL_RCCEx_DisableLSCO

HAL_RCCEx_DisableLSCO

Function name

void HAL_RCCEx_DisableLSCO (void)

Function description

Disable the Low Speed clock output.

Return values

- **None:**

HAL_RCCEx_EnableLSCOinDEEPSTOP

Function name

void HAL_RCCEx_EnableLSCOinDEEPSTOP (uint32_t LSCOSource)

Function description

Enable the Low Speed clock source to output on LSCO pin during STANDBY with retention.

Parameters

- **LSCOSource:** specifies the Low Speed clock source to output. This parameter can be one of the following values:
  - RCC_LSCOSOURCE_NOCLOCK Internal LPMU slow clock source selected as LSCO source
  - RCC_LSCOSOURCE_LSI LSI clock selected as LSCO source
  - RCC_LSCOSOURCE_LSE LSE clock selected as LSCO source

Return values

- **None:**

HAL_RCCEx_TrimOsc

Function name

HAL_StatusTypeDef HAL_RCCEx_TrimOsc (uint32_t OscillatorType)

Function description

28.3 RCCEx Firmware driver defines

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

28.3.1 RCCEx

RCCEx

RCCEx Exported Macros

__HAL_RCC_LSCO_CONFIG

Description:

- Macro to configure the LSCO clock.

Parameters:

- __LSCOCLKSOURCE__: specifies the LSCO clock source. This parameter can be one of the following values:
  - RCC_LSCOSOURCE_NOCLOCK LSCO output disabled
  - RCC_LSCOSOURCE_LSI LSI clock selected as LSCO source
  - RCC_LSCOSOURCE_LSE LSE clock selected as LSCO source

__HAL_RCC_SMPS_DIV_CONFIG

Description:

- Macro to configure the SMPS clock prescaling factor.

Parameters:

- __SMPSCLKDIV__: specifies the division factor for SMPS clock. This parameter can be one of the following values:
  - RCC_SMPSCLK_DIV2 SMPS clock division 2 (SMPS clock is 8 MHz)
  - RCC_SMPSCLK_DIV4 SMPS clock division 4 (SMPS clock is 4 MHz)

Return value:

- None

__HAL_RCC_GET_SMPS_DIV

This parameter can be one of the following values:

__HAL_RCC_SPI3I2S_CLK_CONFIG

Description:

- Macro to configure the SPI3_I2S clock.

Parameters:

- __SPI3I2S_CLKSOURCE__: specifies the SPI3I2S clock source. This parameter can be one of the following values:
  - RCC_SPI3I2S_CLKSOURCE_16M
  - RCC_SPI3I2S_CLKSOURCE_32M
  - RCC_SPI3I2S_CLKSOURCE_64M

Return value:

- None

__HAL_RCC_GET_SPI3I2S_CLK_CONFIG

Description:

- Macro to get the SPI3_I2S clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_SPI3I2S_CLKSOURCE_16M
  - RCC_SPI3I2S_CLKSOURCE_32M
  - RCC_SPI3I2S_CLKSOURCE_64M

__HAL_RCC_LPUART1_CLK_CONFIG

Description:

- Macro to configure the LPUART1 clock.

Parameters:

- __LPUART1_CLKSOURCE__: specifies the LPUART1 clock source. This parameter can be one of the following values:
  - RCC_LPUART1_CLKSOURCE_16M
  - RCC_LPUART1_CLKSOURCE_LSE

Return value:

- None

__HAL_RCC_GET_LPUART1_CLK_CONFIG

Description:

- Macro to get the LPUART1 clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_LPUART1_CLKSOURCE_16M
  - RCC_LPUART1_CLKSOURCE_LSE

__HAL_RCC_RTC_WDG_BLEWKUP_CLK_CONFIG

Description:

- Macro to configure the RTC, WDG and BLEWKUP clock source.

Parameters:

- __RTC_WDG_BLEWKUP_CLKSOURCE__: specifies the RTC, WDG and BLEWKUP clock source. This parameter can be one of the following values:
  - RCC_RTC_WDG_BLEWKUP_CLKSOURCE_LSE
  - RCC_RTC_WDG_BLEWKUP_CLKSOURCE_LSI
  - RCC_RTC_WDG_BLEWKUP_CLKSOURCE_HSI64M_DIV2048

Return value:

- None

__HAL_RCC_GET_RTC_WDG_BLEWKUP_CLK_CONFIG

Description:

- Macro to get the RTC, WDG and BLEWKUP clock source.

Return value:

- The: clock source can be one of the following values:
  - RCC_RTC_WDG_BLEWKUP_CLKSOURCE_LSE
  - RCC_RTC_WDG_BLEWKUP_CLKSOURCE_LSI
  - RCC_RTC_WDG_BLEWKUP_CLKSOURCE_HSI64M_DIV2048

LPUART1 Clock Source

RCC_LPUART1_CLKSOURCE_16M

LPUART1 16 MHz clock source selection

RCC_LPUART1_CLKSOURCE_LSE

LPUART1 LSE clock source selection

Periph Clock Selection

RCC_PERIPHCLK_RF

RADIO IP Peripheral Clock Selection

RCC_PERIPHCLK_SMPS

SMPS Peripheral Clock Selection

RCC_PERIPHCLK_SPI3_I2S

SPI3 I2S Peripheral Clock Selection

RCC_PERIPHCLK_LPUART1

LPUART1 Peripheral Clock Selection

RCC_PERIPHCLK_RTC_WDG_BLEWKUP

RTC WDG SUBG BLE_WKUP Peripheral Clock Selection

RTC, WDG, BLEWKUP Clock Source

RCC_RTC_WDG_BLEWKUP_CLKSOURCE_LSE

RTC, WDG and BLEWKUP LSE clock source selection

RCC_RTC_WDG_BLEWKUP_CLKSOURCE_LSI

RTC, WDG and BLEWKUP LSI clock source selection

RCC_RTC_WDG_BLEWKUP_CLKSOURCE_HSI64M_DIV2048

RTC, WDG and BLEWKUP 32K clock source selection

SMPS clock prescaling factor

RCC_SMPSCLK_DIV2

SMPS clock division 2 (SMPS clock is 8 MHz if ANADIV = 1)

RCC_SMPSCLK_DIV4

SMPS clock division 4 (SMPS clock is 4 MHz if ANADIV = 1)

SPI3_I2S Clock Source

RCC_SPI3I2S_CLKSOURCE_16M

SPI3 I2S 16 MHz clock source selection

RCC_SPI3I2S_CLKSOURCE_32M

SPI3 I2S 32 MHz clock source selection

29 HAL RNG Generic Driver

29.1 RNG Firmware driver registers structures

29.1.1 RNG_InitTypeDef

RNG_InitTypeDef is defined in the stm32wb0x_hal_rng.h

Data Fields

- ***uint32_t ClockErrorDetection***

Field Documentation

- ***uint32_t RNG_InitTypeDef::ClockErrorDetection***
  CED Clock error detection not use for WB09 product

29.1.2 RNG_HandleTypeDef

RNG_HandleTypeDef is defined in the stm32wb0x_hal_rng.h

Data Fields

- ***RNG_TypeDef * Instance***
- ***RNG_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RNG_StateTypeDef State***
- ***__IO uint32_t ErrorCode***
- ***uint32_t RandomNumber***

Field Documentation

- ***RNG_TypeDef* RNG_HandleTypeDef::Instance***
  Register base address
- ***RNG_InitTypeDef RNG_HandleTypeDef::Init***
  RNG configuration parameters
- ***HAL_LockTypeDef RNG_HandleTypeDef::Lock***
  RNG locking object
- ***__IO HAL_RNG_StateTypeDef RNG_HandleTypeDef::State***
  RNG communication state
- ***__IO uint32_t RNG_HandleTypeDef::ErrorCode***
  RNG Error code
- ***uint32_t RNG_HandleTypeDef::RandomNumber***
  Last Generated RNG Data

29.2 RNG Firmware driver API description

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

29.2.1 How to use this driver

The RNG HAL driver can be used as follows:

1. Enable the RNG controller clock using __HAL_RCC_RNG_CLK_ENABLE() macro in HAL_RNG_MspInit().
2. Activate the RNG peripheral using HAL_RNG_Init() function.
3. Wait until the 16/32 bit Random Number Generator contains a valid random data using polling mode.
4. Get the 16/32 bit random number using HAL_RNG_GenerateRandomNumber() function.

29.2.2 Initialization and configuration functions

This section provides functions allowing to:

- Initialize the RNG according to the specified parameters in the RNG_InitTypeDef and create the associated handle

- DeInitialize the RNG peripheral
- Initialize the RNG MSP
- DeInitialize RNG MSP

This section contains the following APIs:

- *HAL_RNG_Init()*
- *HAL_RNG_DeInit()*
- *HAL_RNG_MspInit()*
- *HAL_RNG_MspDeInit()*

29.2.3 Peripheral Control functions

This section provides functions allowing to:

- Get the 16/32 bit Random number

This section contains the following APIs:

- *HAL_RNG_GenerateRandomNumber()*
- *HAL_RNG_ReadLastRandomNumber()*

29.2.4 Peripheral State functions

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

This section contains the following APIs:

- *HAL_RNG_GetState()*
- *HAL_RNG_GetError()*

29.2.5 Detailed description of functions

HAL_RNG_Init

Function name

HAL_StatusTypeDef HAL_RNG_Init (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG peripheral and creates the associated handle.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_DeInit

Function name

HAL_StatusTypeDef HAL_RNG_DeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG peripheral.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** status

HAL_RNG_MspInit

Function name

void HAL_RNG_MspInit (RNG_HandleTypeDef * hrng)

Function description

Initializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_MspDeInit

Function name

void HAL_RNG_MspDeInit (RNG_HandleTypeDef * hrng)

Function description

DeInitializes the RNG MSP.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **None:**

HAL_RNG_GenerateRandomNumber

Function name

HAL_StatusTypeDef HAL_RNG_GenerateRandomNumber (RNG_HandleTypeDef * hrng, uint32_t * random_number)

Function description

Generates a 16/32-bit random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.
- **random_number:** random 16/32bit pointer to generated random number variable if successful.

Return values

- **HAL:** status

Notes

- Each time the random number data is read the RNG_FLAG_DRDY flag is automatically cleared.

HAL_RNG_ReadLastRandomNumber

Function name

uint32_t HAL_RNG_ReadLastRandomNumber (const RNG_HandleTypeDef * hrng)

Function description

Read latest generated random number.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **random:** value

HAL_RNG_GetState

Function name

HAL_RNG_StateTypeDef HAL_RNG_GetState (const RNG_HandleTypeDef * hrng)

Function description

Returns the RNG state.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure that contains the configuration information for RNG.

Return values

- **HAL:** state

HAL_RNG_GetError

Function name

uint32_t HAL_RNG_GetError (const RNG_HandleTypeDef * hrng)

Function description

Return the RNG handle error code.

Parameters

- **hrng:** pointer to a RNG_HandleTypeDef structure.

Return values

- **RNG:** Error Code

29.3 RNG Firmware driver defines

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

29.3.1 RNG

RNG

RNG Error Definition

HAL_RNG_ERROR_NONE

No error

HAL_RNG_ERROR_TIMEOUT

Timeout error

HAL_RNG_ERROR_BUSY

Busy error

RNG Flag definition

RNG_FLAG_DRDY

New Random Data Ready

RNG Exported Macros

__HAL_RNG_RESET_HANDLE_STATE

Description:
- Reset RNG handle state.

Parameters:
- __HANDLE__: RNG Handle

Return value:
- None

__HAL_RNG_ENABLE

Description:
- Enables the RNG peripheral.

Parameters:
- __HANDLE__: RNG Handle

Return value:
- None

__HAL_RNG_DISABLE

Description:
- Disables the RNG peripheral.

Parameters:
- __HANDLE__: RNG Handle

Return value:
- None

__HAL_RNG_GET_FLAG

Description:
- Check the selected RNG flag status.

Parameters:
- __HANDLE__: RNG Handle
- __FLAG__: RNG flag This parameter can be one of the following values:
  – RNG_FLAG_DRDY: Data ready
  – RNG_FLAG_CECS: Clock error current status
  – RNG_FLAG_SECS: Seed error current status

Return value:
- The: new state of __FLAG__ (SET or RESET).

30 HAL RTC Generic Driver

30.1 RTC Firmware driver registers structures

30.1.1 RTC_InitTypeDef

RTC_InitTypeDef is defined in the stm32wb0x_hal_rtc.h

Data Fields

- ***uint32_t HourFormat***
- ***uint32_t AsynchPrediv***
- ***uint32_t SynchPrediv***
- ***uint32_t OutPut***
- ***uint32_t OutPutPolarity***

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 = 0x0000 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 ***RTC_Output_selection_Definitions***
- ***uint32_t RTC_InitTypeDef::OutPutPolarity***
  Specifies the polarity of the output signal. This parameter can be a value of ***RTC_Output_Polarity_Definitions***

30.1.2 RTC_TimeTypeDef

RTC_TimeTypeDef is defined in the stm32wb0x_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 prescaler factor value (PREDIV_S) This parameter corresponds to a time unit range between [0-1] Second with [1 Sec / SecondFraction +1] granularity. This field will be used only by HAL_RTC_GetTime function
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
  This interface is deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions

30.1.3 RTC_DateTypeDef

RTC_DateTypeDef is defined in the stm32wb0x_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 stm32wb0x_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 stm32wb0x_hal_rtc.h

Data Fields

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***
- ***void(* AlarmAEventCallback***
- ***void(* WakeUpTimerEventCallback***
- ***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::WakeUpTimerEventCallback)(struct __RTC_HandleTypeDef *hrtc)***
  RTC WakeUpTimer 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 RTC and Backup Domain Operating Condition

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

1. The RTC
2. The LSE oscillator

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins are available:

1. PC14 and PC15 can be used as either GPIO or LSE pins
2. PC13 can be used as a GPIO or as the RTC_AF1 pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

1. PC14 and PC15 can be used as LSE pins only

2. PC13 can be used as the RTC_AF1 pin

30.2.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.

30.2.3 Backup Domain Access

After reset, the backup domain (RTC registers and RTC backup data registers) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the __HAL_RCC_PWR_CLK_ENABLE() macro.
- Enable access to RTC domain using the HAL_PWR_EnableBkUpAccess() function.
- Select the RTC clock source using the __HAL_RCC_RTC_CONFIG() macro.
- Enable RTC Clock using the __HAL_RCC_RTC_ENABLE() macro.

30.2.4 How to use this driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the HAL_RTC_SetTime() and HAL_RTC_SetDate() functions.
- To read the RTC Calendar, use the HAL_RTC_GetTime() and HAL_RTC_GetDate() functions.
- To manage the RTC summer or winter time change, use the following functions:
  – HAL_RTC_DST_Add1Hour() or HAL_RTC_DST_Sub1Hour to add or subtract 1 hour from the calendar time.
  – HAL_RTC_DST_SetStoreOperation() or HAL_RTC_DST_ClearStoreOperation to memorize whether the time change has been performed or not.

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.

30.2.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarm (Alarm A), and RTC wakeup. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

Callback registration

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. This is the recommended configuration in order to optimize memory/code consumption footprint/performances.

The compilation define USE_HAL_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.
- WakeUpTimerEventCallback : RTC WakeUpTimer Event callback.
- MspInitCallback : RTC MspInit callback.
- MspDeInitCallback : RTC MspDeInit callback.

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.
- WakeUpTimerEventCallback : RTC WakeUpTimer 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 for MspInit() and MspDeInit() that can be registered/unregistered in HAL_RTC_STATE_READY or HAL_RTC_STATE_RESET state. Thus registered (user) MspInit()/MspDeInit() 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() functions.

30.2.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
   – A 7-bit asynchronous prescaler and a 15-bit synchronous prescaler.
   – When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- *HAL_RTC_Init()*
- *HAL_RTC_DeInit()*
- *HAL_RTC_RegisterCallback()*
- *HAL_RTC_UnRegisterCallback()*
- *HAL_RTC_MspInit()*
- *HAL_RTC_MspDeInit()*

30.2.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- *HAL_RTC_SetTime()*
- *HAL_RTC_GetTime()*
- *HAL_RTC_SetDate()*
- *HAL_RTC_GetDate()*

30.2.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- *HAL_RTC_SetAlarm()*
- *HAL_RTC_SetAlarm_IT()*
- *HAL_RTC_DeactivateAlarm()*
- *HAL_RTC_GetAlarm()*
- *HAL_RTC_AlarmIRQHandler()*
- *HAL_RTC_AlarmAEventCallback()*
- *HAL_RTC_PollForAlarmAEvent()*

30.2.9 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization
- Manage RTC Summer or Winter time change

This section contains the following APIs:

- *HAL_RTC_WaitForSynchro()*
- *HAL_RTC_DST_Add1Hour()*
- *HAL_RTC_DST_Sub1Hour()*
- *HAL_RTC_DST_SetStoreOperation()*
- *HAL_RTC_DST_ClearStoreOperation()*
- *HAL_RTC_DST_ReadStoreOperation()*

30.2.10 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- *HAL_RTC_GetState()*

30.2.11 Detailed description of functions

HAL_RTC_Init

Function name

HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)

Function description

Initializes the RTC peripheral.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** status

HAL_RTC_DeInit

Function name

HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitializes the RTC peripheral.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

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

Initializes the RTC MSP.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTC_MspDeInit

Function name

void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)

Function description

DeInitializes the RTC MSP.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

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

Registers a User RTC Callback To be used instead of the weak predefined callback.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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_WAKEUPTIMER_EVENT_CB_ID Wakeup Timer 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

Unregisters an RTC Callback RTC callback is redirected to the weak predefined callback.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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_WAKEUPTIMER_EVENT_CB_ID Wakeup Timer 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

Sets RTC current time.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Notes

- DayLightSaving and StoreOperation interfaces are deprecated. To manage Daylight Saving Time, please use HAL_RTC_DST_xxx functions.

HAL_RTC_GetTime

Function name

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

Function description

Gets RTC current time.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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

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

Sets RTC current date.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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 (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)

Function description

Gets RTC current date.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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 to ensure consistency between the time and date values.

HAL_RTC_SetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Sets the specified RTC Alarm.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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.

HAL_RTC_SetAlarm_IT

Function name

HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)

Function description

Sets the specified RTC Alarm with Interrupt.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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 application must ensure that the RTC interrupt line is configured via HAL SYSCFG services and the PWR_WAKEUP_PIN_RTC bit is set.
- 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.

HAL_RTC_DeactivateAlarm

Function name

HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)

Function description

Deactivates the specified RTC Alarm.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **Alarm:** Specifies the Alarm. This parameter can be:
  - RTC_ALARM_A: Alarm A

Return values

- **HAL:** status

HAL_RTC_GetAlarm

Function name

HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)

Function description

Gets the RTC Alarm value and masks.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **sAlarm:** Pointer to Date structure
- **Alarm:** Specifies the Alarm. This parameter can be:
  - RTC_ALARM_A: Alarm A
- **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

Handles Alarm interrupt request.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTC_PollForAlarmAEvent

Function name

HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handles Alarm A Polling request.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTC_WaitForSynchro

Function name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro (RTC_HandleTypeDef * hrtc)

Function description

Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

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_DST_Add1Hour

Function name

void HAL_RTC_DST_Add1Hour (RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, adds one hour to the calendar in one single operation without going through the initialization procedure.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTC_DST_Sub1Hour

Function name

void HAL_RTC_DST_Sub1Hour (RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, subtracts one hour from the calendar in one single operation without going through the initialization procedure.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTC_DST_SetStoreOperation

Function name

void HAL_RTC_DST_SetStoreOperation (RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, sets the store operation bit.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

Notes

- It can be used by the software in order to memorize the DST status.

HAL_RTC_DST_ClearStoreOperation

Function name

void HAL_RTC_DST_ClearStoreOperation (RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, clears the store operation bit.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTC_DST_ReadStoreOperation

Function name

uint32_t HAL_RTC_DST_ReadStoreOperation (RTC_HandleTypeDef * hrtc)

Function description

Daylight Saving Time, reads the store operation bit.

Parameters

- **hrtc:** RTC handle

Return values

- **operation:** see RTC_StoreOperation_Definitions

HAL_RTC_GetState

Function name

HAL_RTCStateTypeDef HAL_RTC_GetState (RTC_HandleTypeDef * hrtc)

Function description

Returns the RTC state.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** state

RTC_EnterInitMode

Function name

HAL_StatusTypeDef RTC_EnterInitMode (RTC_HandleTypeDef * hrtc)

Function description

Enters the RTC Initialization mode.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

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

Exits the RTC Initialization mode.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** status

RTC_ByteToBcd2

Function name

uint8_t RTC_ByteToBcd2 (uint8_t number)

Function description

Converts a 2-digit number from decimal to BCD format.

Parameters

- **number:** decimal-formatted number (from 0 to 99) to be converted

Return values

- **Converted:** byte

RTC_Bcd2ToByte

Function name

uint8_t RTC_Bcd2ToByte (uint8_t number)

Function description

Converts a 2-digit number from BCD to decimal format.

Parameters

- **number:** BCD-formatted number (from 00 to 99) 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 Alarm Date WeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

RTC Alarm Mask 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 SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.

RTC_ALARMSUBSECONDMASK_SS14_1

SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_2

SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_3

SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_4

SS[14:5] are don't care in Alarm comparison. Only SS[4:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_5

SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_6

SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_7

SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_8

SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_9

SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_10

SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_11

SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_12

SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14_13

SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared.

RTC_ALARMSUBSECONDMASK_SS14

SS[14:0] are compared and must match to activate alarm.

RTC_ALARMSUBSECONDMASK_NONE

RTC AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

RTC DayLight Saving 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__: specifies the 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_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:
  - RTC_IT_ALRA: Alarm A interrupt

Return value:

- None

__HAL_RTC_ALARM_DISABLE_IT

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be:
  - 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 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 Alarm's flag status.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag to check. This parameter can be:
  – RTC_FLAG_ALRAF: Alarm A interrupt flag
  – RTC_FLAG_ALRAWF: Alarm A 'write allowed' flag

Return value:
- None

__HAL_RTC_ALARM_CLEAR_FLAG

Description:
- Clear the RTC Alarm's pending flags.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm flag to be cleared. This parameter can be:
  – RTC_FLAG_ALRAF

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

RTC Flags Definitions

RTC_FLAG_RECALPF

Recalibration pending flag

RTC_FLAG_WUTF

Wakeup timer event flag

RTC_FLAG_ALRAF

Alarm A event flag

RTC_FLAG_INITF

RTC in initialization mode flag

RTC_FLAG_RSF

Register synchronization flag

RTC_FLAG_INITS

RTC initialization status flag

RTC_FLAG_SHPF

Shift operation pending flag

RTC_FLAG_WUTWF

WUTR register write allowance flag

RTC_FLAG_ALRAWF

ALRMAR register write allowance flag

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_WUT

Enable Wakeup timer Interrupt

RTC_IT_ALRA

Enable Alarm A Interrupt

RTC Private macros to check input parameters

IS_RTC_HOUR_FORMAT

IS_RTC_OUTPUT

IS_RTC_OUTPUT_POL

IS_RTC_ASYNCH_PREDIV

IS_RTC_SYNCH_PREDIV

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

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

RTC Month Date Definitions (in BCD format)

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 Polarity Definitions

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

RTC Output Selection Definitions

RTC_OUTPUT_DISABLE

RTC_OUTPUT_ALARMA

RTC_OUTPUT_WAKEUP

RTC Store Operation 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.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEx_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer in interrupt mode using the HAL_RTCEx_SetWakeUpTimer_IT() function.
- To read the RTC Wakeup Counter register, use the HAL_RTCEx_GetWakeUpTimer() function.

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEx_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEx_BKUPRead() function.

Smooth Digital Calibration configuration

- RTC frequency can be digitally calibrated with a resolution of about 0.954 ppm with a range from -487.1 ppm to +488.5 ppm. The correction of the frequency is performed using a series of small adjustments (adding and/or subtracting individual RTCCLK pulses).
- The smooth digital calibration is performed during a cycle of about 2^20 RTCCLK pulses (or 32 seconds) when the input frequency is 32,768 Hz. This cycle is maintained by a 20-bit counter clocked by RTCCLK.
- The smooth calibration register (RTC_CALR) specifies the number of RTCCLK clock cycles to be masked during the 32-second cycle.
- To configure the RTC Smooth Digital Calibration value and the corresponding calibration cycle period (32s,16s and 8s) use the HAL_RTCEx_SetSmoothCalib() function.

31.1.2 RTC Wakeup functions

This section provides functions allowing to configure Wakeup feature

This section contains the following APIs:

- *HAL_RTCEx_SetWakeUpTimer()*
- *HAL_RTCEx_SetWakeUpTimer_IT()*
- *HAL_RTCEx_DeactivateWakeUpTimer()*
- *HAL_RTCEx_GetWakeUpTimer()*
- *HAL_RTCEx_WakeUpTimerIRQHandler()*
- *HAL_RTCEx_WakeUpTimerEventCallback()*
- *HAL_RTCEx_PollForWakeUpTimerEvent()*

31.1.3 Extended Peripheral Control functions

This subsection provides functions allowing to

- Write a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Set the 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_BKUPWrite()*
- *HAL_RTCEx_BKUPRead()*
- *HAL_RTCEx_SetSmoothCalib()*
- *HAL_RTCEx_SetSynchroShift()*
- *HAL_RTCEx_SetCalibrationOutPut()*
- *HAL_RTCEx_DeactivateCalibrationOutPut()*
- *HAL_RTCEx_EnableBypassShadow()*
- *HAL_RTCEx_DisableBypassShadow()*
- *HAL_RTCEx_SetRefClock()*
- *HAL_RTCEx_DeactivateRefClock()*

31.1.4 Detailed description of functions

HAL_RTCEx_SetWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Sets wakeup timer.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **WakeUpCounter:** Wakeup counter
- **WakeUpClock:** Wakeup clock

Return values

- **HAL:** status

HAL_RTCEx_SetWakeUpTimer_IT

Function name

HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)

Function description

Sets wakeup timer with interrupt.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **WakeUpCounter:** Wakeup counter
- **WakeUpClock:** Wakeup clock

Return values

- **HAL:** status

Notes

- The application must ensure that the RTC interrupt line is configured via HAL SYSCFG services and the PWR_WAKEUP_PIN_RTC bit is set.

HAL_RTCEx_DeactivateWakeUpTimer

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Deactivates wakeup timer counter.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** status

HAL_RTCEx_GetWakeUpTimer

Function name

uint32_t HAL_RTCEx_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)

Function description

Gets wakeup timer counter.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **Counter:** value

HAL_RTCEx_WakeUpTimerIRQHandler

Function name

void HAL_RTCEx_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)

Function description

Handles Wakeup Timer interrupt request.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTCEx_WakeUpTimerEventCallback

Function name

void HAL_RTCEx_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)

Function description

Wakeup Timer callback.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **None:**

HAL_RTCEx_PollForWakeUpTimerEvent

Function name

HAL_StatusTypeDef HAL_RTCEx_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)

Function description

Handles Wakeup Timer Polling.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **Timeout:** Timeout duration

Return values

- **HAL:** status

HAL_RTCEx_BKUPWrite

Function name

void HAL_RTCEx_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified RTC Backup data register.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx (where x can be from 0 to 1) to specify the register.
- **Data:** Data to be written in the specified RTC Backup data register.

Return values

- **None:**

HAL_RTCEx_BKUPRead

Function name

uint32_t HAL_RTCEx_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx (where x can be from 0 to 1) to specify the register.

Return values

- **Read:** value

HAL_RTCEx_SetSmoothCalib

Function name

HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)

Function description

Sets the Smooth calibration parameters.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Configures the Synchronization Shift Control Settings.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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

HAL_RTCEx_SetCalibrationOutPut

Function name

HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)

Function description

Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **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

Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- **HAL:** status

HAL_RTCEx_SetRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_DeactivateRefClock

Function name

HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)

Function description

HAL_RTCEx_EnableBypassShadow

Function name

HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)

Function description

Enables the Bypass Shadow feature.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

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

Disables the Bypass Shadow feature.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

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

RTCEx Backup Registers Definitions

RTC_BKP_DR0

RTC_BKP_DR1

RTCEx 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_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operation's 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: Shift pending flag

Return value:

- None

RTCEx Calib Output Selection Definitions

RTC_CALIBOUTPUT_512HZ

RTC_CALIBOUTPUT_1HZ

Private macros to check input parameters

IS_RTC_BKP

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

IS_RTC_SMOOTH_CALIB_PERIOD

IS_RTC_SMOOTH_CALIB_PLUS

IS_RTC_SMOOTH_CALIB_MINUS

IS_RTC_SHIFT_ADD1S

IS_RTC_SHIFT_SUBFS

IS_RTC_CALIB_OUTPUT

RTCEx Smooth Calib Period Definitions

RTC_SMOOTHCALIB_PERIOD_32SEC

If RTCCLK = 32768 Hz, smooth calibration period is 32s, otherwise 2^20 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_16SEC

If RTCCLK = 32768 Hz, smooth calibration period is 16s, otherwise 2^19 RTCCLK pulses

RTC_SMOOTHCALIB_PERIOD_8SEC

If RTCCLK = 32768 Hz, smooth calibration period is 8s, otherwise 2^18 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 WakeUp Timer

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC Wakeup Timer peripheral.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_ENABLE_IT

Description:

- Enable the RTC Wakeup Timer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wakeup Timer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_WUT: Wakeup Timer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE_IT

Description:

- Disable the RTC Wakeup Timer interrupt.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wakeup Timer interrupt sources to be enabled or disabled. This parameter can be:
  - RTC_IT_WUT: Wakeup Timer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT

Description:

- Check whether the specified RTC Wakeup Timer interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wakeup Timer interrupt to check. This parameter can be:
  - RTC_IT_WUT: Wakeup Timer interrupt

Return value:

- None

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

Description:
- Check whether the specified RTC Wakeup timer interrupt has been enabled or not.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wakeup timer interrupt sources to check. This parameter can be:
  – RTC_IT_WUT: WakeUpTimer interrupt

Return value:
- None

__HAL_RTC_WAKEUPTIMER_GET_FLAG

Description:
- Get the selected RTC Wakeup Timer's flag status.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Wakeup Timer flag to check. This parameter can be:
  – RTC_FLAG_WUTF: Wakeup Timer interrupt flag
  – RTC_FLAG_WUTWF: Wakeup Timer 'write allowed' flag

Return value:
- None

__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG

Description:
- Clear the RTC Wakeup timer's pending flags.

Parameters:
- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Wakeup Timer Flag to clear. This parameter can be:
  – RTC_FLAG_WUTF: Wakeup Timer interrupt Flag

Return value:
- None

RTCEx Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

32 HAL SMARTCARD Generic Driver

32.1 SMARTCARD Firmware driver registers structures

32.1.1 SMARTCARD_InitTypeDef

SMARTCARD_InitTypeDef is defined in the stm32wb0x_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 stm32wb0x_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 stm32wb0x_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 (such as 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 0x0FFFFFFFF.

Return values

- **None:**

HAL_SMARTCARDEx_EnableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableReceiverTimeOut

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableReceiverTimeOut (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the SMARTCARD receiver timeout feature.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **HAL:** status

HAL_SMARTCARDEx_RxFifoFullCallback

Function name

void HAL_SMARTCARDEx_RxFifoFullCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD RX Fifo full callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_TxFifoEmptyCallback

Function name

void HAL_SMARTCARDEx_TxFifoEmptyCallback (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

SMARTCARD TX Fifo empty callback.

Parameters

- **hsmartcard:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- **None:**

HAL_SMARTCARDEx_EnableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_EnableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Enable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_DisableFifoMode

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_DisableFifoMode (SMARTCARD_HandleTypeDef * hsmartcard)

Function description

Disable the FIFO mode.

Parameters

- **hsmartcard:** SMARTCARD handle.

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetTxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetTxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the TXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** TX FIFO threshold value This parameter can be one of the following values:
  – SMARTCARD_TXFIFO_THRESHOLD_1_8
  – SMARTCARD_TXFIFO_THRESHOLD_1_4
  – SMARTCARD_TXFIFO_THRESHOLD_1_2
  – SMARTCARD_TXFIFO_THRESHOLD_3_4
  – SMARTCARD_TXFIFO_THRESHOLD_7_8
  – SMARTCARD_TXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

HAL_SMARTCARDEx_SetRxFifoThreshold

Function name

HAL_StatusTypeDef HAL_SMARTCARDEx_SetRxFifoThreshold (SMARTCARD_HandleTypeDef * hsmartcard, uint32_t Threshold)

Function description

Set the RXFIFO threshold.

Parameters

- **hsmartcard:** SMARTCARD handle.
- **Threshold:** RX FIFO threshold value This parameter can be one of the following values:
  – SMARTCARD_RXFIFO_THRESHOLD_1_8
  – SMARTCARD_RXFIFO_THRESHOLD_1_4
  – SMARTCARD_RXFIFO_THRESHOLD_1_2
  – SMARTCARD_RXFIFO_THRESHOLD_3_4
  – SMARTCARD_RXFIFO_THRESHOLD_7_8
  – SMARTCARD_RXFIFO_THRESHOLD_8_8

Return values

- **HAL:** status

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 stm32wb0x_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 stm32wb0x_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 STM32WB0x devices contains the following additional features

- Disable or enable Fast Mode Plus

35.1.2 How to use this driver

35.1.3 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()*

35.1.4 Detailed description of functions

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:**

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:**

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_FASTMODEPLUS_PA0

Enable Fast Mode Plus on PA0

SMBUS_FASTMODEPLUS_PA1

Enable Fast Mode Plus on PA1

SMBUS_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

SMBUS_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

36 HAL SPI Generic Driver

36.1 SPI Firmware driver registers structures

36.1.1 SPI_InitTypeDef

SPI_InitTypeDef is defined in the stm32wb0x_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 stm32wb0x_hal_spi.h

Data Fields

- ***SPI_TypeDef * Instance***
- ***SPI_InitTypeDef Init***
- ***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
- ***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()

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, 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, 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, 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, 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, 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, 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 (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 (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 stm32wb0x_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 stm32wb0x_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 stm32wb0x_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 stm32wb0x_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 stm32wb0x_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 stm32wb0x_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 stm32wb0x_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_SlaveConfigTypeDef

TIM_SlaveConfigTypeDef is defined in the stm32wb0x_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

38.1.9 TIM_BreakDeadTimeConfigTypeDef

TIM_BreakDeadTimeConfigTypeDef is defined in the stm32wb0x_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 BreakAFMode***
- ***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::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::AutomaticOutput***
  TIM Automatic Output Enable state, This parameter can be a value of ***TIM_AOE_Bit_Set_Reset***

38.1.10 __TIM_HandleTypeDef

__TIM_HandleTypeDef is defined in the stm32wb0x_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(* PeriodElapsedCallback***
- ***void(* PeriodElapsedHalfCpltCallback***
- ***void(* TriggerCallback***
- ***void(* IC_CaptureCallback***
- ***void(* IC_CaptureHalfCpltCallback***
- ***void(* OC_DelayElapsedCallback***
- ***void(* PWM_PulseFinishedCallback***
- ***void(* PWM_PulseFinishedHalfCpltCallback***
- ***void(* ErrorCallback***
- ***void(* CommutationCallback***
- ***void(* BreakCallback***

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[5]***
  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[4]***
  TIM channel operation state

- ***__IO HAL_TIM_ChannelStateTypeDef __TIM_HandleTypeDef::ChannelNState[1]***
  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::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::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::BreakCallback)(struct __TIM_HandleTypeDef *htim)***
  TIM Break 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.
- PeriodElapsedCallback : TIM Period Elapsed Callback.
- PeriodElapsedHalfCpltCallback : TIM Period Elapsed half complete Callback.
- TriggerCallback : TIM Trigger 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 (when supported).

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_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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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_OR1
  - TIM_DMABASE_AF1
- **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
- **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_OR1
  - TIM_DMABASE_AF1
- **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
- **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_OR1
  – TIM_DMABASE_AF1
- **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
- **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_OR1
  – TIM_DMABASE_AF1
- **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
- **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 are relevant only for timer instances supporting a break input.

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_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_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_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_BREAK_CB_ID Break Callback ID #if defined(TIM_BDTR_BK2E)
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID #endif
- **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_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_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_BREAK_CB_ID Break Callback ID #if defined(TIM_BDTR_BK2E)
  - HAL_TIM_BREAK2_CB_ID Break2 Callback ID #endif

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_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_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_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 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

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_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 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_OR1

TIM_DMABASE_AF1

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 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 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

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

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: 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_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 (*) Value not defined for all devices

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: 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_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 (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices
- __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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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 (*) (*) Value not defined for all devices

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_COM

Commutation interrupt flag

TIM_FLAG_TRIGGER

Trigger interrupt flag

TIM_FLAG_BREAK

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 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 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_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 TIMEx_BreakInputConfigTypeDef

TIMEx_BreakInputConfigTypeDef is defined in the stm32wb0x_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.

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:
2. 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()

39.2.3 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()*

39.2.4 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()*

39.2.5 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()*

39.2.6 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.
- Enable or disable channel grouping.

This section contains the following APIs:

- *HAL_TIMEx_ConfigCommutEvent()*

- *HAL_TIMEx_ConfigCommutEvent_IT()*
- *HAL_TIMEx_ConfigBreakDeadTime()*
- *HAL_TIMEx_ConfigBreakInput()*
- *HAL_TIMEx_RemapConfig()*
- *HAL_TIMEx_DisarmBreakInput()*
- *HAL_TIMEx_ReArmBreakInput()*

39.2.7 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_BreakCallback()*

39.2.8 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_GetChannelNState()*

39.2.9 Detailed description of functions

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:** Unused parameter (kept to avoid compatibility break).
- **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

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:** Unused parameter (kept to avoid compatibility break).
- **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

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_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. This parameter can be one of the following values:
  – TIM_TIM17_TI1_GPIO
  – TIM_TIM17_TI1_LCO
  – TIM_TIM17_TI1_MCO

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_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_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

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_TIM17_TI1_GPIO

TIM17 TI1 is connected to GPIO

TIM_TIM17_TI1_LC0

TIM17 TI1 is connected to LCO

TIM_TIM17_TI1_MCO

TIM17 TI1 is connected to MCO

40 HAL UART Generic Driver

40.1 UART Firmware driver registers structures

40.1.1 UART_InitTypeDef

UART_InitTypeDef is defined in the stm32wb0x_hal_uart.h

Data Fields

- ***uint32_t BaudRate***
- ***uint32_t WordLength***
- ***uint32_t StopBits***
- ***uint32_t Parity***
- ***uint32_t Mode***
- ***uint32_t HwFlowCtl***
- ***uint32_t OverSampling***
- ***uint32_t OneBitSampling***
- ***uint32_t ClockPrescaler***

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
  This member configures the UART communication baud rate. The baud rate register is computed using the following formula: LPUART: Baud Rate Register = ((256 * lpuart_ker_ckpres) / ((huart->Init.BaudRate))) where lpuart_ker_ck_pres is the UART input clock divided by a prescaler UART:
  – If oversampling is 16 or in LIN mode, Baud Rate Register = ((uart_ker_ckpres) / ((huart->Init.BaudRate)))
  – If oversampling is 8, Baud Rate Register[15:4] = ((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[15:4] Baud Rate Register[3] = 0 Baud Rate Register[2:0] = (((2 * uart_ker_ckpres) / ((huart->Init.BaudRate)))[3:0]) >> 1 where uart_ker_ck_pres is the UART input clock divided by a prescaler
- ***uint32_t UART_InitTypeDef::WordLength***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of ***UARTEx_Word_Length***.
- ***uint32_t UART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of ***UART_Stop_Bits***.
- ***uint32_t UART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of ***UART_Parity***
  Note:
  – When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
  Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of ***UART_Mode***.
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of ***UART_Hardware_Flow_Control***.
- ***uint32_t UART_InitTypeDef::OverSampling***
  Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to f_PCLK/8). This parameter can be a value of ***UART_Over_Sampling***.
- ***uint32_t UART_InitTypeDef::OneBitSampling***
  Specifies whether a single sample or three samples' majority vote is selected. Selecting the single sample method increases the receiver tolerance to clock deviations. This parameter can be a value of ***UART_OneBit_Sampling***.
- ***uint32_t UART_InitTypeDef::ClockPrescaler***
  Specifies the prescaler value used to divide the UART clock source. This parameter can be a value of ***UART_ClockPrescaler***.

40.1.2 UART_AdvFeatureInitTypeDef

UART_AdvFeatureInitTypeDef is defined in the stm32wb0x_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 stm32wb0x_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 0x0FFFFFFFF.

Return values

- **None:**

HAL_UART_EnableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_EnableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_UART_DisableReceiverTimeout

Function name

HAL_StatusTypeDef HAL_UART_DisableReceiverTimeout (UART_HandleTypeDef * huart)

Function description

Disable the UART receiver timeout feature.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- **HAL:** status

HAL_LIN_SendBreak

Function name

HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)

Function description

Transmit break characters.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_EnableMuteMode (UART_HandleTypeDef * huart)

Function description

Enable UART in mute mode (does not mean UART enters mute mode; to enter mute mode, HAL_MultiProcessor_EnterMuteMode() API must be called).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_DisableMuteMode

Function name

HAL_StatusTypeDef HAL_MultiProcessor_DisableMuteMode (UART_HandleTypeDef * huart)

Function description

Disable UART mute mode (does not mean the UART actually exits mute mode as it may not have been in mute mode at this very moment).

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_MultiProcessor_EnterMuteMode

Function name

void HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)

Function description

Enter UART mute mode (means UART actually enters mute mode).

Parameters

- **huart:** UART handle.

Return values

- **None:**

Notes

- To exit from mute mode, HAL_MultiProcessor_DisableMuteMode() API must be called.

HAL_HalfDuplex_EnableTransmitter

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)

Function description

Enable the UART transmitter and disable the UART receiver.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

HAL_HalfDuplex_EnableReceiver

Function name

HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)

Function description

Enable the UART receiver and disable the UART transmitter.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status.

HAL_UART_GetState

Function name

HAL_UART_StateTypeDef HAL_UART_GetState (const UART_HandleTypeDef * huart)

Function description

Return the UART handle state.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **HAL:** state

HAL_UART_GetError

Function name

uint32_t HAL_UART_GetError (const UART_HandleTypeDef * huart)

Function description

Return the UART handle error code.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.

Return values

- **UART:** Error Code

UART_InitCallbacksToDefault

Function name

void UART_InitCallbacksToDefault (UART_HandleTypeDef * huart)

Function description

Initialize the callbacks to their default values.

Parameters

- **huart:** UART handle.

Return values

- **none:**

UART_SetConfig

Function name

HAL_StatusTypeDef UART_SetConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_CheckIdleState

Function name

HAL_StatusTypeDef UART_CheckIdleState (UART_HandleTypeDef * huart)

Function description

Check the UART Idle State.

Parameters

- **huart:** UART handle.

Return values

- **HAL:** status

UART_WaitOnFlagUntilTimeout

Function name

HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout (UART_HandleTypeDef * huart, uint32_t Flag, FlagStatus Status, uint32_t Tickstart, uint32_t Timeout)

Function description

This function handles UART Communication Timeout.

Parameters

- **huart:** UART handle.
- **Flag:** Specifies the UART flag to check
- **Status:** The actual Flag status (SET or RESET)
- **Tickstart:** Tick start value
- **Timeout:** Timeout duration

Return values

- **HAL:** status

UART_AdvFeatureConfig

Function name

void UART_AdvFeatureConfig (UART_HandleTypeDef * huart)

Function description

Configure the UART peripheral advanced features.

Parameters

- **huart:** UART handle.

Return values

- **None:**

UART_Start_Receive_IT

Function name

HAL_StatusTypeDef UART_Start_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in interrupt mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in Interrupt mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

UART_Start_Receive_DMA

Function name

HAL_StatusTypeDef UART_Start_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)

Function description

Start Receive operation in DMA mode.

Parameters

- **huart:** UART handle.
- **pData:** Pointer to data buffer (u8 or u16 data elements).
- **Size:** Amount of data elements (u8 or u16) to be received.

Return values

- **HAL:** status

Notes

- This function could be called by all HAL UART API providing reception in DMA mode.
- When calling this function, parameters validity is considered as already checked, i.e. Rx State, buffer address, ... UART Handle is assumed as Locked.

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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  – UART_CLEAR_WUF Wake Up from stop mode Clear Flag #endif #endif

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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  - UART_FLAG_WUF Wake up from stop mode flag #endif #endif
  - 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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  – UART_IT_WUF Wakeup from stop mode interrupt #endif #endif
  – 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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  - UART_IT_WUF Wakeup from stop mode interrupt #endif #endif
  - 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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  - UART_IT_WUF Wakeup from stop mode interrupt #endif #endif
  - 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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  - UART_IT_WUF Wakeup from stop mode interrupt #endif #endif
  - 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 #if defined(USART_CR1_UESM) #if defined(USART_CR3_WUFIE)
  - UART_CLEAR_WUF Wake Up from stop mode Clear Flag #endif #endif

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 stm32wb0x_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. #if defined(USART_CR1_UESM)
- 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 #endif
- 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 stm32wb0x_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 stm32wb0x_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 LL ADC Generic Driver

44.1 ADC Firmware driver registers structures

44.1.1 LL_ADC_InitTypeDef

LL_ADC_InitTypeDef is defined in the stm32wb0x_ll_adc.h

Data Fields

- ***FunctionalState ContinuousConvMode***
- ***uint32_t SequenceLength***
- ***uint32_t SamplingMode***
- ***uint32_t SampleRate***
- ***uint32_t Overrun***

Field Documentation

- ***FunctionalState LL_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. This feature can be modified afterwards using unitary function **LL_ADC_ContinuousModeEnable()** or **LL_ADC_ContinuousModeDisable()**.
- ***uint32_t LL_ADC_InitTypeDef::SequenceLength***
  Specify the length of the conversion sequence. This parameter must be a number between Min_Data = 1 and Max_Data = 16. This feature can be modified afterwards using unitary function **LL_ADC_SetSequenceLength()**.
- ***uint32_t LL_ADC_InitTypeDef::SamplingMode***
  Specifies the input sampling mode. This parameter can be a value of ***ADC_LL_SAMPLING_METHOD***. This feature can be modified afterwards using unitary function **LL_ADC_SetInputSamplingMode()**.
- ***uint32_t LL_ADC_InitTypeDef::SampleRate***
  Specify the ADC sample rate. This parameter can be a value of ***ADC_LL_SAMPLE_RATE***. This feature can be modified afterwards using unitary function **LL_ADC_SetSampleRate()**.
- ***uint32_t LL_ADC_InitTypeDef::Overrun***
  Specifies the overrung policy applied to the data. This parameter can be a value of ***ADC_LL_OVERRUN_CONFIG***. This feature can be modified afterwards using unitary function **LL_ADC_SetOverrunDS()**.

44.1.2 LL_ADC_VoltRangeInitTypeDef

LL_ADC_VoltRangeInitTypeDef is defined in the stm32wb0x_ll_adc.h

Data Fields

- ***uint32_t InputVinm0_Vinp0Vinm0***
- ***uint32_t InputVinm1_Vinp1Vinm1***
- ***uint32_t InputVinm2_Vinp2Vinm2***
- ***uint32_t InputVinm3_Vinp3Vinm3***
- ***uint32_t InputVinp0***
- ***uint32_t InputVinp1***
- ***uint32_t InputVinp2***
- ***uint32_t InputVinp3***

Field Documentation

- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinm0_Vinp0Vinm0***
  Set the input voltage range for single VINM0 or differential (VINP0 - VINM0). This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.

- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinm1_Vinp1Vinm1***
  Set the input voltage range for single VINM1 or differential (VINP1 - VINM1). This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.
- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinm2_Vinp2Vinm2***
  Set the input voltage range for single VINM2 or differential (VINP2 - VINM2). This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.
- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinm3_Vinp3Vinm3***
  Set the input voltage range for single VINM3 or differential (VINP3 - VINM3). This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.
- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinp0***
  Set the input voltage range for single VINP0. This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.
- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinp1***
  Set the input voltage range for single VINP1. This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.
- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinp2***
  Set the input voltage range for single VINP2. This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.
- ***uint32_t LL_ADC_VoltRangeInitTypeDef::InputVinp3***
  Set the input voltage range for single VINP3. This parameter can be a value of ***ADC_LL_INPUT_VOLTAGE_RANGE*** This feature can be modified afterwards using unitary function **LL_ADC_SetChannelVoltageRange()**.

44.1.3 LL_ADC_SequenceInitTypeDef

LL_ADC_SequenceInitTypeDef is defined in the stm32wb0x_ll_adc.h

Data Fields

- ***uint32_t ChannelForSeq0***
- ***uint32_t ChannelForSeq1***
- ***uint32_t ChannelForSeq2***
- ***uint32_t ChannelForSeq3***
- ***uint32_t ChannelForSeq4***
- ***uint32_t ChannelForSeq5***
- ***uint32_t ChannelForSeq6***
- ***uint32_t ChannelForSeq7***
- ***uint32_t ChannelForSeq8***
- ***uint32_t ChannelForSeq9***
- ***uint32_t ChannelForSeq10***
- ***uint32_t ChannelForSeq11***
- ***uint32_t ChannelForSeq12***
- ***uint32_t ChannelForSeq13***
- ***uint32_t ChannelForSeq14***
- ***uint32_t ChannelForSeq15***

Field Documentation

- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq0***
  Set the channel number code for the 1st conversion of the sequence. This parameter can be a value of ***ADC_LL_CHANNEL*** This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.

- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq1***
  Set the channel number code for the 2nd conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq2***
  Set the channel number code for the 3rd conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq3***
  Set the channel number code for the 4th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq4***
  Set the channel number code for the 5th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq5***
  Set the channel number code for the 6th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq6***
  Set the channel number code for the 7th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq7***
  Set the channel number code for the 8th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq8***
  Set the channel number code for the 9th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq9***
  Set the channel number code for the 10th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq10***
  Set the channel number code for the 11th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq11***
  Set the channel number code for the 12th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq12***
  Set the channel number code for the 13th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq13***
  Set the channel number code for the 14th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.
- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq14***
  Set the channel number code for the 15th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.

- ***uint32_t LL_ADC_SequenceInitTypeDef::ChannelForSeq15***
  Set the channel number code for the 16th conversion of the sequence. This parameter can be a value of *ADC_LL_CHANNEL* This feature can be modified afterwards using unitary function **LL_ADC_SetSequencerRanks()**.

44.2 ADC Firmware driver API description

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

44.2.1 Detailed description of functions

LL_ADC_SetInputSamplingMode

Function name

__STATIC_INLINE void LL_ADC_SetInputSamplingMode (ADC_TypeDef * ADCx, uint32_t SamplingMode)

Function description

Select the input sampling method.

Parameters

- **ADCx:** ADC instance
- **SamplingMode:** This parameter can be one of the following values:
  – LL_ADC_SAMPLING_AT_START
  – LL_ADC_SAMPLING_AT_END

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF ADC_CONT_1V2 LL_ADC_SetInputSamplingMode

LL_ADC_GetInputSamplingMode

Function name

__STATIC_INLINE uint32_t LL_ADC_GetInputSamplingMode (const ADC_TypeDef * ADCx)

Function description

Get input sampling mode.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_SAMPLING_AT_START
  – LL_ADC_SAMPLING_AT_END

Reference Manual to LL API cross reference:

- CONF ADC_CONT_1V2 LL_ADC_GetInputSamplingMode

LL_ADC_InvertOutputDiffModeEnable

Function name

__STATIC_INLINE void LL_ADC_InvertOutputDiffModeEnable (ADC_TypeDef * ADCx)

Function description

Enable the inversion of the ADC data output bits (1's complement) when a differential input is connected to the ADC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF BIT_INVERT_DIFF LL_ADC_InvertOutputDiffModeEnable

LL_ADC_InvertOutputDiffModeDisable

Function name

__STATIC_INLINE void LL_ADC_InvertOutputDiffModeDisable (ADC_TypeDef * ADCx)

Function description

Disable the inversion of the ADC data output bits (1's complement) when a differential input is connected to the ADC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF BIT_INVERT_DIFF LL_ADC_InvertOutputDiffModeDisable

LL_ADC_IsInvertOutputDiffModeEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInvertOutputDiffModeEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the inversion of the ADC data output bits (1's complement) when a differential input is connected to the ADC is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** inversion is not enabled, 1: inversion is enabled.

Reference Manual to LL API cross reference:

- CONF BIT_INVERT_DIFF LL_ADC_IsInvertOutputDiffModeEnabled

LL_ADC_InvertOutputSingleNegModeEnable

Function name

__STATIC_INLINE void LL_ADC_InvertOutputSingleNegModeEnable (ADC_TypeDef * ADCx)

Function description

Enable the inversion of the ADC data output bits (1's complement) when a single negative input is connected to the ADC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF BIT_INVERT_SN LL_ADC_InvertOutputSingleNegModeEnable

LL_ADC_InvertOutputSingleNegModeDisable

Function name

__STATIC_INLINE void LL_ADC_InvertOutputSingleNegModeDisable (ADC_TypeDef * ADCx)

Function description

Disable the inversion of the ADC data output bits (1's complement) when a single negative input is connected to the ADC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF BIT_INVERT_SN LL_ADC_InvertOutputSingleNegModeDisable

LL_ADC_IsInvertOutputSingleNegModeEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInvertOutputSingleNegModeEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the inversion of the ADC data output bits (1's complement) when a single negative input is connected to the ADC is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** inversion is not enabled, 1: inversion is enabled.

Reference Manual to LL API cross reference:

- CONF BIT_INVERT_SN LL_ADC_IsInvertOutputSingleNegModeEnabled

LL_ADC_SetOverrunDS

Function name

__STATIC_INLINE void LL_ADC_SetOverrunDS (ADC_TypeDef * ADCx, uint32_t Overrun)

Function description

Select the overrun configuration for the output data of the Down Sampler (DS).

Parameters

- **ADCx:** ADC instance
- **Overrun:** This parameter can be one of the following values:
  - LL_ADC_NEW_DATA_IS_LOST
  - LL_ADC_NEW_DATA_IS_KEPT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF OVR_DS_CFG LL_ADC_SetOverrunDS

LL_ADC_GetOverrunDS

Function name

__STATIC_INLINE uint32_t LL_ADC_GetOverrunDS (const ADC_TypeDef * ADCx)

Function description

Get the overrun configuration for the output data of the Down Sampler (DS).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  – LL_ADC_NEW_DATA_IS_LOST
  – LL_ADC_NEW_DATA_IS_KEPT

Reference Manual to LL API cross reference:

- CONF OVR_DS_CFG LL_ADC_GetOverrunDS

LL_ADC_DMAModeDSEnable

Function name

__STATIC_INLINE void LL_ADC_DMAModeDSEnable (ADC_TypeDef * ADCx)

Function description

Enable the DMA mode for the Down Sampler (DS) data path.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF DMA_DS_ENA LL_ADC_DMAModeDSEnable

LL_ADC_DMAModeDSDisable

Function name

__STATIC_INLINE void LL_ADC_DMAModeDSDisable (ADC_TypeDef * ADCx)

Function description

Disable the DMA mode for the Down Sampler (DS) data path.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF DMA_DS_ENA LL_ADC_DMAModeDSDisable

LL_ADC_IsDMAModeDSEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsDMAModeDSEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the DMA mode for the Down Sampler (DS) data path is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** DMA mode is not enabled, 1: DMA mode is enabled.

Reference Manual to LL API cross reference:

- CONF DMA_DS_ENA LL_ADC_IsDMAModeDSEnabled

LL_ADC_SetSampleRate

Function name

__STATIC_INLINE void LL_ADC_SetSampleRate (ADC_TypeDef * ADCx, uint32_t SampleRate)

Function description

Select the ADC conversion rate.

Parameters

- **ADCx:** ADC instance
- **SampleRate:** This parameter can be one of the following values:
  – LL_ADC_SAMPLE_RATE_16
  – LL_ADC_SAMPLE_RATE_20
  – LL_ADC_SAMPLE_RATE_24
  – LL_ADC_SAMPLE_RATE_28
  – LL_ADC_SAMPLE_RATE_32 (1)
  – LL_ADC_SAMPLE_RATE_36 (1)
  – LL_ADC_SAMPLE_RATE_40 (1)
  – LL_ADC_SAMPLE_RATE_44 (1)
  – LL_ADC_SAMPLE_RATE_48 (1)
  – LL_ADC_SAMPLE_RATE_52 (1)
  – LL_ADC_SAMPLE_RATE_56 (1)
  – LL_ADC_SAMPLE_RATE_60 (1)
  – LL_ADC_SAMPLE_RATE_64 (1)
  – LL_ADC_SAMPLE_RATE_68 (1)
  – LL_ADC_SAMPLE_RATE_72 (1)
  – LL_ADC_SAMPLE_RATE_76 (1)
  – LL_ADC_SAMPLE_RATE_80 (1)
  – LL_ADC_SAMPLE_RATE_84 (1)
  – LL_ADC_SAMPLE_RATE_88 (1)
  – LL_ADC_SAMPLE_RATE_92 (1)
  – LL_ADC_SAMPLE_RATE_96 (1)
  – LL_ADC_SAMPLE_RATE_100 (1)
  – LL_ADC_SAMPLE_RATE_104 (1)
  – LL_ADC_SAMPLE_RATE_108 (1)
  – LL_ADC_SAMPLE_RATE_112 (1)
  – LL_ADC_SAMPLE_RATE_116 (1)
  – LL_ADC_SAMPLE_RATE_120 (1)
  – LL_ADC_SAMPLE_RATE_124 (1)
  – LL_ADC_SAMPLE_RATE_128 (1)
  – LL_ADC_SAMPLE_RATE_132 (1)
  – LL_ADC_SAMPLE_RATE_136 (1)
  – LL_ADC_SAMPLE_RATE_140 (1)

  (1) Parameter only available on STM32WB09 and STM32WB05

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF SAMPLE_RATE LL_ADC_SetSampleRate

LL_ADC_GetSampleRate

Function name

__STATIC_INLINE uint32_t LL_ADC_GetSampleRate (const ADC_TypeDef * ADCx)

Function description

Get the ADC conversion rate.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_SAMPLE_RATE_16
  - LL_ADC_SAMPLE_RATE_20
  - LL_ADC_SAMPLE_RATE_24
  - LL_ADC_SAMPLE_RATE_28
  - LL_ADC_SAMPLE_RATE_32 (1)
  - LL_ADC_SAMPLE_RATE_36 (1)
  - LL_ADC_SAMPLE_RATE_40 (1)
  - LL_ADC_SAMPLE_RATE_44 (1)
  - LL_ADC_SAMPLE_RATE_48 (1)
  - LL_ADC_SAMPLE_RATE_52 (1)
  - LL_ADC_SAMPLE_RATE_56 (1)
  - LL_ADC_SAMPLE_RATE_60 (1)
  - LL_ADC_SAMPLE_RATE_64 (1)
  - LL_ADC_SAMPLE_RATE_68 (1)
  - LL_ADC_SAMPLE_RATE_72 (1)
  - LL_ADC_SAMPLE_RATE_76 (1)
  - LL_ADC_SAMPLE_RATE_80 (1)
  - LL_ADC_SAMPLE_RATE_84 (1)
  - LL_ADC_SAMPLE_RATE_88 (1)
  - LL_ADC_SAMPLE_RATE_92 (1)
  - LL_ADC_SAMPLE_RATE_96 (1)
  - LL_ADC_SAMPLE_RATE_100 (1)
  - LL_ADC_SAMPLE_RATE_104 (1)
  - LL_ADC_SAMPLE_RATE_108 (1)
  - LL_ADC_SAMPLE_RATE_112 (1)
  - LL_ADC_SAMPLE_RATE_116 (1)
  - LL_ADC_SAMPLE_RATE_120 (1)
  - LL_ADC_SAMPLE_RATE_124 (1)
  - LL_ADC_SAMPLE_RATE_128 (1)
  - LL_ADC_SAMPLE_RATE_132 (1)
  - LL_ADC_SAMPLE_RATE_136 (1)
  - LL_ADC_SAMPLE_RATE_140 (1)

  (1) Parameter only available on STM32WB09 and STM32WB05

Reference Manual to LL API cross reference:

- CONF SAMPLE_RATE LL_ADC_GetSampleRate

LL_ADC_SMPSSyncEnable

Function name

__STATIC_INLINE void LL_ADC_SMPSSyncEnable (ADC_TypeDef * ADCx)

Function description

Enable the synchronization of the ADC start conversion with a pulse generated by the SMPS.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This bitfield must be 0 when the ADC clock is 32 MHz or when the SMPS is not used.

Reference Manual to LL API cross reference:

- CONF SMPS_SYNCHRO_ENA LL_ADC_SMPSSyncEnable

LL_ADC_SMPSSyncDisable

Function name

__STATIC_INLINE void LL_ADC_SMPSSyncDisable (ADC_TypeDef * ADCx)

Function description

Disable the synchronization of the ADC start conversion with a pulse generated by the SMPS.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This bitfield must be 0 when the ADC clock is 32 MHz or when the SMPS is not used.

Reference Manual to LL API cross reference:

- CONF SMPS_SYNCHRO_ENA LL_ADC_SMPSSyncDisable

LL_ADC_IsSMPSSyncEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsSMPSSyncEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the the synchronization of the ADC start conversion with a pulse generated by the SMPS is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** Synchronization with SMPS is not enabled, 1: Synchronization with SMPS is enabled.

Notes

- This bitfield must be 0 when the ADC clock is 32 MHz or when the SMPS is not used.

Reference Manual to LL API cross reference:

- CONF SMPS_SYNCHRO_ENA LL_ADC_IsSMPSSyncEnabled

LL_ADC_SetSequenceLength

Function name

__STATIC_INLINE void LL_ADC_SetSequenceLength (ADC_TypeDef * ADCx, uint32_t SequenceLength)

Function description

Set the number of conversion in a sequence starting from SEQ0.

Parameters

- **ADCx:** ADC instance
- **SequenceLength:** This parameter must be a number between Min_Data = 1 and Max_Data = 16.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF SEQ_LEN LL_ADC_SetSequenceLength

LL_ADC_GetSequenceLength

Function name

__STATIC_INLINE uint32_t LL_ADC_GetSequenceLength (const ADC_TypeDef * ADCx)

Function description

Get the number of conversion in a sequence starting from SEQ0.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value is a number between Min_Data = 1 and Max_Data = 16.

Reference Manual to LL API cross reference:

- CONF SEQ_LEN LL_ADC_GetSequenceLength

LL_ADC_SequenceModeEnable

Function name

__STATIC_INLINE void LL_ADC_SequenceModeEnable (ADC_TypeDef * ADCx)

Function description

Enable the ADC conversion in sequence mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF SEQUENCE LL_ADC_SequenceModeEnable

LL_ADC_SequenceModeDisable

Function name

__STATIC_INLINE void LL_ADC_SequenceModeDisable (ADC_TypeDef * ADCx)

Function description

Disable the ADC conversion in sequence mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF SEQUENCE LL_ADC_SequenceModeDisable

LL_ADC_IsSequenceModeEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsSequenceModeEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the ADC conversion in sequence mode is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** Sequence mode is not enabled, 1: Sequence mode is enabled.

Reference Manual to LL API cross reference:

- CONF SEQUENCE LL_ADC_IsSequenceModeEnabled

LL_ADC_ContinuousModeEnable

Function name

__STATIC_INLINE void LL_ADC_ContinuousModeEnable (ADC_TypeDef * ADCx)

Function description

Enable the ADC continuous conversion mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- Stop the conversion can be made by sets the STOP_OP_MODE bit. Call the API LL_ADC_StopConversion().

Reference Manual to LL API cross reference:

- CONF CONT LL_ADC_ContinuousModeEnable

LL_ADC_ContinuousModeDisable

Function name

__STATIC_INLINE void LL_ADC_ContinuousModeDisable (ADC_TypeDef * ADCx)

Function description

Disable the ADC continuous conversion mode.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONF CONT LL_ADC_ContinuousModeDisable

LL_ADC_IsContinuousModeEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsContinuousModeEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the ADC continuous conversion mode is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** Continuous mode is not enabled, 1: Continuous mode is enabled.

Reference Manual to LL API cross reference:

- CONF CONT LL_ADC_IsContinuousModeEnabled

LL_ADC_EnableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_EnableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Enable the LDO associated to the ADC block.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This bit must not be set on QFN32 package.

Reference Manual to LL API cross reference:

- CTRL ADC_LDO_ENA LL_ADC_EnableInternalRegulator

LL_ADC_DisableInternalRegulator

Function name

__STATIC_INLINE void LL_ADC_DisableInternalRegulator (ADC_TypeDef * ADCx)

Function description

Disable the LDO associated to the ADC block.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This bit must not be set on QFN32 package.

Reference Manual to LL API cross reference:

- CTRL ADC_LDO_ENA LL_ADC_DisableInternalRegulator

LL_ADC_IsInternalRegulatorEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsInternalRegulatorEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the LDO associated to the ADC block is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** LDO is not enabled, 1: LDO is enabled.

Notes

- This bit must not be set on QFN32 package.

Reference Manual to LL API cross reference:

- CTRL ADC_LDO_ENA LL_ADC_IsInternalRegulatorEnabled

LL_ADC_StopConversion

Function name

__STATIC_INLINE void LL_ADC_StopConversion (ADC_TypeDef * ADCx)

Function description

Stop the ongoing ADC conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This bit is set by software and cleared by hardware.
- When the STOP_MODE_OP is set, the user has to wait around 10 us before to start a new ADC conversion (set START_CONV bit).

Reference Manual to LL API cross reference:

- CTRL STOP_OP_MODE LL_ADC_StopConversion

LL_ADC_IsStopConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsStopConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Check if the stop conversion is ongoing.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** stop conversion is not ongoing, 1: stop conversion is ongoing.

Reference Manual to LL API cross reference:

- CTRL STOP_OP_MODE LL_ADC_IsStopConversionOngoing

LL_ADC_StartConversion

Function name

__STATIC_INLINE void LL_ADC_StartConversion (ADC_TypeDef * ADCx)

Function description

Start an ADC conversion.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Notes

- This bit is set by software and cleared by hardware.

Reference Manual to LL API cross reference:

- CTRL START_CONV LL_ADC_StartConversion

LL_ADC_IsConversionOngoing

Function name

__STATIC_INLINE uint32_t LL_ADC_IsConversionOngoing (const ADC_TypeDef * ADCx)

Function description

Check if the ADC conversion is ongoing.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC conversion is not ongoing, 1: ADC conversion is ongoing.

Reference Manual to LL API cross reference:

- CTRL ADC_ON_OFF LL_ADC_IsConversionOngoing

LL_ADC_Enable

Function name

__STATIC_INLINE void LL_ADC_Enable (ADC_TypeDef * ADCx)

Function description

Power on the ADC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTRL ADC_ON_OFF LL_ADC_Enable

LL_ADC_Disable

Function name

__STATIC_INLINE void LL_ADC_Disable (ADC_TypeDef * ADCx)

Function description

Power off the ADC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CTRL ADC_ON_OFF LL_ADC_Disable

LL_ADC_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabled (const ADC_TypeDef * ADCx)

Function description

Check if the ADC is on.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** ADC is off, 1: ADC is on.

Reference Manual to LL API cross reference:

- CTRL ADC_ON_OFF LL_ADC_IsEnabled

LL_ADC_SetChannelVoltageRange

Function name

__STATIC_INLINE void LL_ADC_SetChannelVoltageRange (ADC_TypeDef * ADCx, uint32_t Channel, uint32_t Range)

Function description

Set the input voltage range for selected channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  – LL_ADC_CHANNEL_VINM0
  – LL_ADC_CHANNEL_VINM1
  – LL_ADC_CHANNEL_VINM2
  – LL_ADC_CHANNEL_VINM3
  – LL_ADC_CHANNEL_VINP0
  – LL_ADC_CHANNEL_VINP1
  – LL_ADC_CHANNEL_VINP2
  – LL_ADC_CHANNEL_VINP3
  – LL_ADC_CHANNEL_VINP0_VINM0
  – LL_ADC_CHANNEL_VINP1_VINM1
  – LL_ADC_CHANNEL_VINP2_VINM2
  – LL_ADC_CHANNEL_VINP3_VINM3
  – LL_ADC_CHANNEL_VBAT
  – LL_ADC_CHANNEL_TEMPSENSOR
- **Range:** This parameter can be one of the following values:
  – LL_ADC_VIN_RANGE_1V2
  – LL_ADC_VIN_RANGE_2V4
  – LL_ADC_VIN_RANGE_3V6

Return values

- **None:**

Reference Manual to LL API cross reference:

- SWITCH SE_VIN_0 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_1 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_2 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_3 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_4 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_5 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_6 LL_ADC_SetChannelVoltageRange
- SWITCH SE_VIN_7 LL_ADC_SetChannelVoltageRange

LL_ADC_GetChannelVoltageRange

Function name

__STATIC_INLINE uint32_t LL_ADC_GetChannelVoltageRange (const ADC_TypeDef * ADCx, uint32_t Channel)

Function description

Get the input voltage range for selected channel.

Parameters

- **ADCx:** ADC instance
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VINM0
  - LL_ADC_CHANNEL_VINM1
  - LL_ADC_CHANNEL_VINM2
  - LL_ADC_CHANNEL_VINM3
  - LL_ADC_CHANNEL_VINP0
  - LL_ADC_CHANNEL_VINP1
  - LL_ADC_CHANNEL_VINP2
  - LL_ADC_CHANNEL_VINP3
  - LL_ADC_CHANNEL_VINP0_VINM0
  - LL_ADC_CHANNEL_VINP1_VINM1
  - LL_ADC_CHANNEL_VINP2_VINM2
  - LL_ADC_CHANNEL_VINP3_VINM3
  - LL_ADC_CHANNEL_VBAT
  - LL_ADC_CHANNEL_TEMPSENSOR

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Reference Manual to LL API cross reference:

- SWITCH SE_VIN_0 LL_ADC_GetChannelVoltageRange SWITCH SE_VIN_1 LL_ADC_GetChannelVoltageRange
- SWITCH SE_VIN_2 LL_ADC_GetChannelVoltageRange
- SWITCH SE_VIN_3 LL_ADC_GetChannelVoltageRange
- SWITCH SE_VIN_4 LL_ADC_GetChannelVoltageRange
- SWITCH SE_VIN_5 LL_ADC_GetChannelVoltageRange
- SWITCH SE_VIN_6 LL_ADC_GetChannelVoltageRange
- SWITCH SE_VIN_7 LL_ADC_GetChannelVoltageRange

LL_ADC_ConfigureDSDataOutput

Function name

__STATIC_INLINE void LL_ADC_ConfigureDSDataOutput (ADC_TypeDef * ADCx, uint32_t Width, uint32_t Ratio)

Function description

Configure the width (in bit) and the ratio of the output data from the Down Sampler.

Parameters

- **ADCx:** ADC instance
- **Width:** This parameter can be one of the following values:
  – LL_ADC_DS_DATA_WIDTH_12_BIT (default)
  – LL_ADC_DS_DATA_WIDTH_13_BIT
  – LL_ADC_DS_DATA_WIDTH_14_BIT
  – LL_ADC_DS_DATA_WIDTH_15_BIT
  – LL_ADC_DS_DATA_WIDTH_16_BIT
- **Ratio:** This parameter can be one of the following values:
  – LL_ADC_DS_RATIO_1 (no down sampling, default)
  – LL_ADC_DS_RATIO_2
  – LL_ADC_DS_RATIO_4
  – LL_ADC_DS_RATIO_8
  – LL_ADC_DS_RATIO_16
  – LL_ADC_DS_RATIO_32
  – LL_ADC_DS_RATIO_64
  – LL_ADC_DS_RATIO_128

Return values

- **None:**

LL_ADC_SetDSDataOutputWidth

Function name

__STATIC_INLINE void LL_ADC_SetDSDataOutputWidth (ADC_TypeDef * ADCx, uint32_t Width)

Function description

Set the width (in bit) of the output data from the Down Sampler.

Parameters

- **ADCx:** ADC instance
- **Width:** This parameter can be one of the following values:
  – LL_ADC_DS_DATA_WIDTH_12_BIT (default)
  – LL_ADC_DS_DATA_WIDTH_13_BIT
  – LL_ADC_DS_DATA_WIDTH_14_BIT
  – LL_ADC_DS_DATA_WIDTH_15_BIT
  – LL_ADC_DS_DATA_WIDTH_16_BIT

Return values

- **None:**

Reference Manual to LL API cross reference:

- DS_CONF DS_WIDTH LL_ADC_SetDSDataOutputWidth

LL_ADC_GetDSDataOutputWidth

Function name

__STATIC_INLINE uint32_t LL_ADC_GetDSDataOutputWidth (const ADC_TypeDef * ADCx)

Function description

Get the width (in bit) of the output data from the Down Sampler (DS).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT (default)
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT

Reference Manual to LL API cross reference:

- DS_CONF DS_WIDTH LL_ADC_GetDSDataOutputWidth

LL_ADC_SetDSDataOutputRatio

Function name

__STATIC_INLINE void LL_ADC_SetDSDataOutputRatio (ADC_TypeDef * ADCx, uint32_t Ratio)

Function description

Set the ratio of the Down Sampler (DS).

Parameters

- **ADCx:** ADC instance
- **Ratio:** This parameter can be one of the following values:
  - LL_ADC_DS_RATIO_1 (no down sampling, default)
  - LL_ADC_DS_RATIO_2
  - LL_ADC_DS_RATIO_4
  - LL_ADC_DS_RATIO_8
  - LL_ADC_DS_RATIO_16
  - LL_ADC_DS_RATIO_32
  - LL_ADC_DS_RATIO_64
  - LL_ADC_DS_RATIO_128

Return values

- **None:**

Reference Manual to LL API cross reference:

- DS_CONF DS_RATIO LL_ADC_SetDSDataOutputRatio

LL_ADC_GetDSDataOutputRatio

Function name

__STATIC_INLINE uint32_t LL_ADC_GetDSDataOutputRatio (const ADC_TypeDef * ADCx)

Function description

Get the ratio of the Down Sampler (DS).

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_DS_RATIO_1 (no down sampling, default)
  - LL_ADC_DS_RATIO_2
  - LL_ADC_DS_RATIO_4
  - LL_ADC_DS_RATIO_8
  - LL_ADC_DS_RATIO_16
  - LL_ADC_DS_RATIO_32
  - LL_ADC_DS_RATIO_64
  - LL_ADC_DS_RATIO_128

Reference Manual to LL API cross reference:

- DS_CONF DS_RATIO LL_ADC_GetDSDataOutputRatio

LL_ADC_SetSequencerRanks

Function name

__STATIC_INLINE void LL_ADC_SetSequencerRanks (ADC_TypeDef * ADCx, uint32_t Rank, uint32_t Channel)

Function description

Set ADC sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_RANK_1
  - LL_ADC_RANK_2
  - LL_ADC_RANK_3
  - LL_ADC_RANK_4
  - LL_ADC_RANK_5
  - LL_ADC_RANK_6
  - LL_ADC_RANK_7
  - LL_ADC_RANK_8
  - LL_ADC_RANK_9
  - LL_ADC_RANK_10
  - LL_ADC_RANK_11
  - LL_ADC_RANK_12
  - LL_ADC_RANK_13
  - LL_ADC_RANK_14
  - LL_ADC_RANK_15
  - LL_ADC_RANK_16
- **Channel:** This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VINM0
  - LL_ADC_CHANNEL_VINM1
  - LL_ADC_CHANNEL_VINM2
  - LL_ADC_CHANNEL_VINM3
  - LL_ADC_CHANNEL_VINP0
  - LL_ADC_CHANNEL_VINP1
  - LL_ADC_CHANNEL_VINP2
  - LL_ADC_CHANNEL_VINP3
  - LL_ADC_CHANNEL_VINP0_VINM0
  - LL_ADC_CHANNEL_VINP1_VINM1
  - LL_ADC_CHANNEL_VINP2_VINM2
  - LL_ADC_CHANNEL_VINP3_VINM3
  - LL_ADC_CHANNEL_VBAT
  - LL_ADC_CHANNEL_TEMPSENSOR

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, ADCsequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.

Reference Manual to LL API cross reference:

- SEQ_1 SEQ1 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ1 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ2 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ3 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ4 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ5 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ6 LL_ADC_SetSequencerRanks
- SEQ_1 SEQ7 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ8 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ9 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ10 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ11 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ12 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ13 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ14 LL_ADC_SetSequencerRanks
- SEQ_2 SEQ15 LL_ADC_SetSequencerRanks

LL_ADC_GetSequencerRanks

Function name

__STATIC_INLINE uint32_t LL_ADC_GetSequencerRanks (const ADC_TypeDef * ADCx, uint32_t Rank)

Function description

Get ADC sequence: channel on the selected scan sequence rank.

Parameters

- **ADCx:** ADC instance
- **Rank:** This parameter can be one of the following values:
  - LL_ADC_RANK_1
  - LL_ADC_RANK_2
  - LL_ADC_RANK_3
  - LL_ADC_RANK_4
  - LL_ADC_RANK_5
  - LL_ADC_RANK_6
  - LL_ADC_RANK_7
  - LL_ADC_RANK_8
  - LL_ADC_RANK_9
  - LL_ADC_RANK_10
  - LL_ADC_RANK_11
  - LL_ADC_RANK_12
  - LL_ADC_RANK_13
  - LL_ADC_RANK_14
  - LL_ADC_RANK_15
  - LL_ADC_RANK_16

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CHANNEL_VINM0
  - LL_ADC_CHANNEL_VINM1
  - LL_ADC_CHANNEL_VINM2
  - LL_ADC_CHANNEL_VINM3
  - LL_ADC_CHANNEL_VINP0
  - LL_ADC_CHANNEL_VINP1
  - LL_ADC_CHANNEL_VINP2
  - LL_ADC_CHANNEL_VINP3
  - LL_ADC_CHANNEL_VINP0_VINM0
  - LL_ADC_CHANNEL_VINP1_VINM1
  - LL_ADC_CHANNEL_VINP2_VINM2
  - LL_ADC_CHANNEL_VINP3_VINM3
  - LL_ADC_CHANNEL_VBAT
  - LL_ADC_CHANNEL_TEMPSENSOR

Notes

- On this STM32 series, ADC sequencer is fully configurable: sequencer length and each rank affectation to a channel are configurable. Refer to description of function LL_ADC_SetSequencerLength().
- Depending on devices and packages, some channels may not be available. Refer to device datasheet for channels availability.

Reference Manual to LL API cross reference:

- SEQ_1 SEQ0 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ1 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ2 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ3 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ4 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ5 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ6 LL_ADC_GetSequencerRanks
- SEQ_1 SEQ7 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ8 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ9 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ10 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ11 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ12 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ13 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ14 LL_ADC_GetSequencerRanks
- SEQ_2 SEQ15 LL_ADC_GetSequencerRanks

LL_ADC_ConfigureCalibPoint

Function name

__STATIC_INLINE void LL_ADC_ConfigureCalibPoint (ADC_TypeDef * ADCx, uint32_t Point, uint32_t Gain, uint32_t Offset)

Function description

Configure the gain and the offset of the calibration point 1.

Parameters

- **ADCx:** ADC instance
- **Gain:** the gain of the first calibration point.
- **Offset:** the signed offset of the first calibration point.
- **Point:** This parameter can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4

Return values

- **None:**

LL_ADC_SetCalibPointGain

Function name

__STATIC_INLINE void LL_ADC_SetCalibPointGain (ADC_TypeDef * ADCx, uint32_t Point, uint32_t Gain)

Function description

Set the gain of the calibration point 1.

Parameters

- **ADCx:** ADC instance
- **Gain:** the gain of the first calibration point.
- **Point:** This parameter can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- COMP_1 GAIN1 LL_ADC_SetCalibPointGain

LL_ADC_GetCalibPointGain

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibPointGain (const ADC_TypeDef * ADCx, uint32_t Point)

Function description

Get the gain of the calibration point 1.

Parameters

- **ADCx:** ADC instance
- **Point:** This parameter can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4

Return values

- **Return:** the gain of the first calibration point.

Reference Manual to LL API cross reference:

- COMP_1 GAIN1 LL_ADC_GetCalibPoint1Gain

LL_ADC_SetCalibPointOffset

Function name

__STATIC_INLINE void LL_ADC_SetCalibPointOffset (ADC_TypeDef * ADCx, uint32_t Point, uint8_t Offset)

Function description

Set the offset of the calibration point 1.

Parameters

- **ADCx:** ADC instance
- **Offset:** the signed offset of the first calibration point.
- **Point:** This parameter can be one of the following values:
  – LL_ADC_CALIB_POINT_1
  – LL_ADC_CALIB_POINT_2
  – LL_ADC_CALIB_POINT_3
  – LL_ADC_CALIB_POINT_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- COMP_1 OFFSET1 LL_ADC_SetCalibPoint1Offset

LL_ADC_GetCalibPointOffset

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibPointOffset (const ADC_TypeDef * ADCx, uint32_t Point)

Function description

Get the offset of the calibration point 1.

Parameters

- **ADCx:** ADC instance
- **Point:** This parameter can be one of the following values:
  – LL_ADC_CALIB_POINT_1
  – LL_ADC_CALIB_POINT_2
  – LL_ADC_CALIB_POINT_3
  – LL_ADC_CALIB_POINT_4

Return values

- **Return:** the signed offset of the first calibration point.

Reference Manual to LL API cross reference:

- COMP_1 OFFSET1 LL_ADC_GetCalibPoint1Offset

LL_ADC_SetCalibPointForDiff

Function name

__STATIC_INLINE void LL_ADC_SetCalibPointForDiff (ADC_TypeDef * ADCx, uint32_t Point, uint32_t Range)

Function description

Set the use of a specific calibration point for ADC differential mode.

Parameters

- **ADCx:** ADC instance
- **Point:** This parameter can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4
- **Range:** This parameter can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Return values

- **None:**

Reference Manual to LL API cross reference:

- COMP_SEL ADC_COMP_SEL_OFFSET_GAIN8 LL_ADC_SetCalibPointForDiff

LL_ADC_GetCalibPointForDiff

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibPointForDiff (const ADC_TypeDef * ADCx, uint32_t Range)

Function description

Get what calibration point is used for ADC differential mode.

Parameters

- **ADCx:** ADC instance
- **Range:** This parameter can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4

Reference Manual to LL API cross reference:

- COMP_SEL ADC_COMP_SEL_OFFSET_GAIN8 LL_ADC_GetCalibPointForDiff

LL_ADC_SetCalibPointForSinglePos

Function name

__STATIC_INLINE void LL_ADC_SetCalibPointForSinglePos (ADC_TypeDef * ADCx, uint32_t Point, uint32_t Range)

Function description

Set the use of a specific calibration point for ADC single positive mode.

Parameters

- **ADCx:** ADC instance
- **Point:** This parameter can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4
- **Range:** This parameter can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Return values

- **None:**

Reference Manual to LL API cross reference:

- COMP_SEL ADC_COMP_SEL_OFFSET_GAIN7 LL_ADC_SetCalibPointForSinglePos3V6

LL_ADC_GetCalibPointForSinglePos

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibPointForSinglePos (const ADC_TypeDef * ADCx, uint32_t Range)

Function description

Get what calibration point is used for ADC single positive mode.

Parameters

- **ADCx:** ADC instance
- **Range:** This parameter can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4

Reference Manual to LL API cross reference:

- COMP_SEL ADC_COMP_SEL_OFFSET_GAIN7 LL_ADC_GetCalibPointForSinglePos3V6

LL_ADC_SetCalibPointForSingleNeg

Function name

__STATIC_INLINE void LL_ADC_SetCalibPointForSingleNeg (ADC_TypeDef * ADCx, uint32_t Point, uint32_t Range)

Function description

Set the use of a specific calibration point for ADC single negative mode and battery level detector.

Parameters

- **ADCx:** ADC instance
- **Point:** This parameter can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4
- **Range:** This parameter can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Return values

- **None:**

Reference Manual to LL API cross reference:

- COMP_SEL ADC_COMP_SEL_OFFSET_GAIN6 LL_ADC_SetCalibPointForSingleNeg

LL_ADC_GetCalibPointForSingleNeg

Function name

__STATIC_INLINE uint32_t LL_ADC_GetCalibPointForSingleNeg (const ADC_TypeDef * ADCx, uint32_t Range)

Function description

Get what calibration point is used for ADC single negative mode and battery level detector.

Parameters

- **ADCx:** ADC instance
- **Range:** This parameter can be one of the following values:
  - LL_ADC_VIN_RANGE_1V2
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_3V6

Return values

- **Returned:** value can be one of the following values:
  - LL_ADC_CALIB_POINT_1
  - LL_ADC_CALIB_POINT_2
  - LL_ADC_CALIB_POINT_3
  - LL_ADC_CALIB_POINT_4

Reference Manual to LL API cross reference:

- COMP_SEL ADC_COMP_SEL_OFFSET_GAIN6 LL_ADC_GetCalibPointForSingleNeg

LL_ADC_ConfigureAWDThresholds

Function name

__STATIC_INLINE void LL_ADC_ConfigureAWDThresholds (ADC_TypeDef * ADCx, uint32_t LowThreshold, uint32_t HighThreshold)

Function description

Configure the WatchDoG threshold low and high.

Parameters

- **ADCx:** ADC instance
- **LowThreshold:** This parameter is a 12-bit value.
- **HighThreshold:** This parameter is a 12-bit value.

Return values

- **None:**

LL_ADC_SetAWDThresholdLow

Function name

__STATIC_INLINE void LL_ADC_SetAWDThresholdLow (ADC_TypeDef * ADCx, uint32_t Threshold)

Function description

Set the WatchDoG threshold low.

Parameters

- **ADCx:** ADC instance
- **Threshold:** This parameter is a 12-bit value.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WD_TH ADC_WD_TH_WD_LT LL_ADC_SetAWDThresholdLow

LL_ADC_GetAWDThresholdLow

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDThresholdLow (const ADC_TypeDef * ADCx)

Function description

Get the WatchDoG threshold low.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value of the low threshold.

Reference Manual to LL API cross reference:

- WD_TH ADC_WD_TH_WD_LT LL_ADC_GetAWDThresholdLow

LL_ADC_SetAWDThresholdHigh

Function name

__STATIC_INLINE void LL_ADC_SetAWDThresholdHigh (ADC_TypeDef * ADCx, uint32_t Threshold)

Function description

Set the WatchDoG threshold high.

Parameters

- **ADCx:** ADC instance
- **Threshold:** This parameter is a 12-bit value.

Return values

- **None:**

Reference Manual to LL API cross reference:

- WD_TH ADC_WD_TH_WD_HT LL_ADC_SetAWDThresholdHigh

LL_ADC_GetAWDThresholdHigh

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDThresholdHigh (const ADC_TypeDef * ADCx)

Function description

Get the WatchDoG threshold high.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value of the high threshold.

Reference Manual to LL API cross reference:

- WD_TH ADC_WD_TH_WD_HT LL_ADC_GetAWDThresholdHigh

LL_ADC_SetAWDInputChannels

Function name

__STATIC_INLINE void LL_ADC_SetAWDInputChannels (ADC_TypeDef * ADCx, uint32_t ChannelMask)

Function description

Set the input channels the watchdog must check.

Parameters

- **ADCx:** ADC instance
- **ChannelMask:** This parameter can be a combination of the following values:
  - LL_ADC_AWD_CH_VINM0
  - LL_ADC_AWD_CH_VINM1
  - LL_ADC_AWD_CH_VINM2
  - LL_ADC_AWD_CH_VINM3
  - LL_ADC_AWD_CH_MICROM
  - LL_ADC_AWD_CH_VBAT
  - LL_ADC_AWD_CH_GND_NEG
  - LL_ADC_AWD_CH_VDDA_NEG
  - LL_ADC_AWD_CH_VINP0
  - LL_ADC_AWD_CH_VINP1
  - LL_ADC_AWD_CH_VINP2
  - LL_ADC_AWD_CH_VINP3
  - LL_ADC_AWD_CH_MICROP
  - LL_ADC_AWD_CH_TEMPSENSOR
  - LL_ADC_AWD_CH_GND_POS
  - LL_ADC_AWD_CH_VDDA_POS

Return values

- **None:**

Notes

- ChannelMask can be an OR of the listed parameters.

Reference Manual to LL API cross reference:

- WD_CONF AWD_CHX LL_ADC_SetAWDInputChannels

LL_ADC_GetAWDInputChannels

Function name

__STATIC_INLINE uint32_t LL_ADC_GetAWDInputChannels (const ADC_TypeDef * ADCx)

Function description

Get the input channels the watchdog must check.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value can be a combination of the following values:
  - LL_ADC_AWD_CH_VINM0
  - LL_ADC_AWD_CH_VINM1
  - LL_ADC_AWD_CH_VINM2
  - LL_ADC_AWD_CH_VINM3
  - LL_ADC_AWD_CH_MICROM
  - LL_ADC_AWD_CH_VBAT
  - LL_ADC_AWD_CH_GND_NEG
  - LL_ADC_AWD_CH_VDDA_NEG
  - LL_ADC_AWD_CH_VINP0
  - LL_ADC_AWD_CH_VINP1
  - LL_ADC_AWD_CH_VINP2
  - LL_ADC_AWD_CH_VINP3
  - LL_ADC_AWD_CH_MICROP
  - LL_ADC_AWD_CH_TEMPSENSOR
  - LL_ADC_AWD_CH_GND_POS
  - LL_ADC_AWD_CH_VDDA_POS

Notes

- The returned value can be an OR of the listed parameters.

Reference Manual to LL API cross reference:

- WD_CONF AWD_CHX LL_ADC_GetAWDInputChannels

LL_ADC_DSGetOutputData

Function name

__STATIC_INLINE uint32_t LL_ADC_DSGetOutputData (const ADC_TypeDef * ADCx)

Function description

Get the 16-bit output data from the Down Sampler (DS).

Parameters

- **ADCx:** ADC instance

Return values

- **The:** output data from DS.

Reference Manual to LL API cross reference:

- DS_DATAOUT DS_DATA LL_ADC_DSGetOutputData

LL_ADC_GetActiveFlags

Function name

__STATIC_INLINE uint32_t LL_ADC_GetActiveFlags (ADC_TypeDef * ADCx)

Function description

Get all the flags status.

Parameters

- **ADCx:** ADC instance

Return values

- **All:** the status flags of the register IRQ_STATUS. The value is a combination of the following values:
  – LL_ADC_IRQ_FLAG_OVRFL
  – LL_ADC_IRQ_FLAG_OVRDF
  – LL_ADC_IRQ_FLAG_EODF
  – LL_ADC_IRQ_FLAG_OVRDS
  – LL_ADC_IRQ_FLAG_AWD1
  – LL_ADC_IRQ_FLAG_EOC
  – LL_ADC_IRQ_FLAG_EOS
  – LL_ADC_IRQ_FLAG_EODS

LL_ADC_ClearActiveFlags

Function name

__STATIC_INLINE void LL_ADC_ClearActiveFlags (ADC_TypeDef * ADCx, uint32_t FlagsMask)

Function description

Clear all the flags status.

Parameters

- **ADCx:** ADC instance
- **FlagsMask:** This parameter can be a combination of the following values:
  – LL_ADC_IRQ_FLAG_OVRFL
  – LL_ADC_IRQ_FLAG_OVRDF
  – LL_ADC_IRQ_FLAG_EODF
  – LL_ADC_IRQ_FLAG_OVRDS
  – LL_ADC_IRQ_FLAG_AWD1
  – LL_ADC_IRQ_FLAG_EOC
  – LL_ADC_IRQ_FLAG_EOS
  – LL_ADC_IRQ_FLAG_EODS

Return values

- **None.:**

LL_ADC_IsActiveFlag

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag (const ADC_TypeDef * ADCx, uint32_t FlagsMask)

Function description

Get if the flags status is set.

Parameters

- **ADCx:** ADC instance
- **FlagsMask:** This parameter can be one of the following values:
  - LL_ADC_IRQ_FLAG_OVRFL
  - LL_ADC_IRQ_FLAG_OVRDF
  - LL_ADC_IRQ_FLAG_EODF
  - LL_ADC_IRQ_FLAG_OVRDS
  - LL_ADC_IRQ_FLAG_AWD1
  - LL_ADC_IRQ_FLAG_EOC
  - LL_ADC_IRQ_FLAG_EOS
  - LL_ADC_IRQ_FLAG_EODS

Return values

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

LL_ADC_IsActiveFlag_OVRDS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_OVRDS (const ADC_TypeDef * ADCx)

Function description

Get the status of the flag OVR_DS.

Parameters

- **ADCx:** ADC instance

Return values

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

Reference Manual to LL API cross reference:

- IRQ_STATUS OVR_DS_IRQ LL_ADC_IsActiveFlag_OVRDS

LL_ADC_ClearFlag_OVRDS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_OVRDS (ADC_TypeDef * ADCx)

Function description

Clear the flag OVR_DS.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_STATUS OVR_DS_IRQ LL_ADC_ClearFlag_OVRDS

LL_ADC_IsActiveFlag_AWD

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_AWD (const ADC_TypeDef * ADCx)

Function description

Get the status of the flag AWD.

Parameters

- **ADCx:** ADC instance

Return values

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

Reference Manual to LL API cross reference:

- IRQ_STATUS AWD_IRQ LL_ADC_IsActiveFlag_AWD

LL_ADC_ClearFlag_AWD

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_AWD (ADC_TypeDef * ADCx)

Function description

Clear the flag AWD.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_STATUS AWD_IRQ LL_ADC_ClearFlag_AWD

LL_ADC_IsActiveFlag_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOS (const ADC_TypeDef * ADCx)

Function description

Get the status of the flag EOS.

Parameters

- **ADCx:** ADC instance

Return values

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

Reference Manual to LL API cross reference:

- IRQ_STATUS EOS_IRQ LL_ADC_IsActiveFlag_EOS

LL_ADC_ClearFlag_EOS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOS (ADC_TypeDef * ADCx)

Function description

Clear the flag EOS.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_STATUS EOS_IRQ LL_ADC_ClearFlag_EOS

LL_ADC_IsActiveFlag_EODS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EODS (const ADC_TypeDef * ADCx)

Function description

Get the status of the flag EODS.

Parameters

- **ADCx:** ADC instance

Return values

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

Reference Manual to LL API cross reference:

- IRQ_STATUS EODS_IRQ LL_ADC_IsActiveFlag_EODS

LL_ADC_ClearFlag_EODS

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EODS (ADC_TypeDef * ADCx)

Function description

Clear the flag EODS.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_STATUS EODS_IRQ LL_ADC_ClearFlag_EODS

LL_ADC_IsActiveFlag_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsActiveFlag_EOC (const ADC_TypeDef * ADCx)

Function description

Get the status of the flag EOC.

Parameters

- **ADCx:** ADC instance

Return values

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

Reference Manual to LL API cross reference:

- IRQ_STATUS EOC_IRQ LL_ADC_IsActiveFlag_EOC

LL_ADC_ClearFlag_EOC

Function name

__STATIC_INLINE void LL_ADC_ClearFlag_EOC (ADC_TypeDef * ADCx)

Function description

Clear the flag EOC.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_STATUS EOC_IRQ LL_ADC_ClearFlag_EOC

LL_ADC_EnableIT

Function name

__STATIC_INLINE void LL_ADC_EnableIT (ADC_TypeDef * ADCx, uint32_t IrqMask)

Function description

Enable the interrupts according to the interrupt mask passed as parameter.

Parameters

- **ADCx:** ADC instance
- **IrqMask:** This parameter can be a combination of the following values:
  – LL_ADC_IRQ_EN_OVRFL
  – LL_ADC_IRQ_EN_OVRDF
  – LL_ADC_IRQ_EN_EODF
  – LL_ADC_IRQ_EN_OVRDS
  – LL_ADC_IRQ_EN_AWD1
  – LL_ADC_IRQ_EN_EOS
  – LL_ADC_IRQ_EN_EODS

Return values

- **None.:**

LL_ADC_DisableIT

Function name

__STATIC_INLINE void LL_ADC_DisableIT (ADC_TypeDef * ADCx, uint32_t IrqMask)

Function description

Disable the interrupts according to the interrupt mask passed as parameter.

Parameters

- **ADCx:** ADC instance
- **IrqMask:** This parameter can be a combination of the following values:
  – LL_ADC_IRQ_EN_OVRFL
  – LL_ADC_IRQ_EN_OVRDF
  – LL_ADC_IRQ_EN_EODF
  – LL_ADC_IRQ_EN_OVRDS
  – LL_ADC_IRQ_EN_AWD1
  – LL_ADC_IRQ_EN_EOS
  – LL_ADC_IRQ_EN_EODS

Return values

- **None.:**

LL_ADC_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT (const ADC_TypeDef * ADCx, uint32_t IrqMask)

Function description

Get if the specific flag is enabled or not.

Parameters

- **ADCx:** ADC instance
- **IrqMask:** This parameter can be one of the following values:
  - LL_ADC_IRQ_FLAG_OVRFL
  - LL_ADC_IRQ_FLAG_OVRDF
  - LL_ADC_IRQ_FLAG_EODF
  - LL_ADC_IRQ_FLAG_OVRDS
  - LL_ADC_IRQ_FLAG_AWD1
  - LL_ADC_IRQ_FLAG_EOC
  - LL_ADC_IRQ_FLAG_EOS
  - LL_ADC_IRQ_FLAG_EODS

Return values

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

LL_ADC_EnableIT_DSOVR

Function name

__STATIC_INLINE void LL_ADC_EnableIT_DSOVR (ADC_TypeDef * ADCx)

Function description

Enable the Down Sampler overrun interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE OVR_DS_IRQ_ENA LL_ADC_EnableIT_DSOVR

LL_ADC_DisableIT_DSOVR

Function name

__STATIC_INLINE void LL_ADC_DisableIT_DSOVR (ADC_TypeDef * ADCx)

Function description

Disable the Down Sampler overrun interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE OVR_DS_IRQ_ENA LL_ADC_DisableIT_DSOVR

LL_ADC_IsEnabledIT_DSOVR

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_DSOVR (const ADC_TypeDef * ADCx)

Function description

Check if the Down Sampler overrun interrupt is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** The interrupt is not enabled, 1: The interrupt is enabled.

Reference Manual to LL API cross reference:

- IRQ_ENABLE OVR_DS_IRQ_ENA LL_ADC_IsEnabledIT_DSOVR

LL_ADC_EnableIT_AWD

Function name

__STATIC_INLINE void LL_ADC_EnableIT_AWD (ADC_TypeDef * ADCx)

Function description

Enable the watchdog event interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE AWD_IRQ_ENA LL_ADC_EnableIT_AWD

LL_ADC_DisableIT_AWD

Function name

__STATIC_INLINE void LL_ADC_DisableIT_AWD (ADC_TypeDef * ADCx)

Function description

Disable the watchdog event interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE AWD_IRQ_ENA LL_ADC_DisableIT_AWD

LL_ADC_IsEnabledIT_AWD

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_AWD (const ADC_TypeDef * ADCx)

Function description

Check if the watchdog event interrupt is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** The interrupt is not enabled, 1: The interrupt is enabled.

Reference Manual to LL API cross reference:

- IRQ_ENABLE AWD_IRQ_ENA LL_ADC_IsEnabledIT_AWD

LL_ADC_EnableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOS (ADC_TypeDef * ADCx)

Function description

Enable the End Of a Sequence of conversion interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE EOS_IRQ_ENA LL_ADC_EnableIT_EOS

LL_ADC_DisableIT_EOS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOS (ADC_TypeDef * ADCx)

Function description

Disable the End Of a Sequence of conversion interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE EOS_IRQ_ENA LL_ADC_DisableIT_EOS

LL_ADC_IsEnabledIT_EOS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOS (const ADC_TypeDef * ADCx)

Function description

Check if the End Of a Sequence of conversion interrupt is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** The interrupt is not enabled, 1: The interrupt is enabled.

Reference Manual to LL API cross reference:

- IRQ_ENABLE EOS_IRQ_ENA LL_ADC_IsEnabledIT_EOS

LL_ADC_EnableIT_EODS

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EODS (ADC_TypeDef * ADCx)

Function description

Enable the End Of a Down Sampler conversion interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE EODS_IRQ_ENA LL_ADC_EnableIT_EODS

LL_ADC_DisableIT_EODS

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EODS (ADC_TypeDef * ADCx)

Function description

Disable the End Of a Down Sampler conversion interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE EODS_IRQ_ENA LL_ADC_DisableIT_EODS

LL_ADC_IsEnabledIT_EODS

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EODS (const ADC_TypeDef * ADCx)

Function description

Check if the End Of a Down Sampler conversion interrupt is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** The interrupt is not enabled, 1: The interrupt is enabled.

Reference Manual to LL API cross reference:

- IRQ_ENABLE EODS_IRQ_ENA LL_ADC_IsEnabledIT_EODS

LL_ADC_EnableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_EnableIT_EOC (ADC_TypeDef * ADCx)

Function description

Enable the End Of Conversion interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE EOC_IRQ_ENA LL_ADC_EnableIT_EOC

LL_ADC_DisableIT_EOC

Function name

__STATIC_INLINE void LL_ADC_DisableIT_EOC (ADC_TypeDef * ADCx)

Function description

Disable the End Of Conversion interrupt.

Parameters

- **ADCx:** ADC instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IRQ_ENABLE EOC_IRQ_ENA LL_ADC_DisableIT_EOC

LL_ADC_IsEnabledIT_EOC

Function name

__STATIC_INLINE uint32_t LL_ADC_IsEnabledIT_EOC (const ADC_TypeDef * ADCx)

Function description

Check if the End Of Conversion interrupt is enabled.

Parameters

- **ADCx:** ADC instance

Return values

- **0:** The interrupt is not enabled, 1: The interrupt is enabled.

Reference Manual to LL API cross reference:

- IRQ_ENABLE EOC_IRQ_ENA LL_ADC_IsEnabledIT_EOC

LL_ADC_SetADCLDODelay

Function name

__STATIC_INLINE void LL_ADC_SetADCLDODelay (ADC_TypeDef * ADCx, uint32_t Delay)

Function description

Set the duration of the waiting time between the ADC_LDO enable and the ADC_ON, to let time to the LDO to stabilize itself before starting a conversion The time unit is 4 us.

Parameters

- **ADCx:** ADC instance
- **Delay:** This parameter is a 8-bit value.

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMER_CONF ADC_LDO_DELAY LL_ADC_SetADCLDODelay

LL_ADC_GetADCLDODelay

Function name

__STATIC_INLINE uint32_t LL_ADC_GetADCLDODelay (const ADC_TypeDef * ADCx)

Function description

Get the duration of the waiting time between the ADC_LDO enable and the ADC_ON, to let time to the LDO to stabilize itself before starting a conversion The time unit is 4 us.

Parameters

- **ADCx:** ADC instance

Return values

- **Returned:** value is a 8-bit value.

Reference Manual to LL API cross reference:

- TIMER_CONF ADC_LDO_DELAY LL_ADC_GetADCLDODelay

LL_ADC_GET_CALIB_GAIN_FOR_VINPX_3V6

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINPX_3V6 (void)

Function description

Return the calibration value for the gain of single ended positive input at range 3.6 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINPX_3V6

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINPX_3V6 (void)

Function description

Return the calibration value for the offset of single ended positive input at range 3.6 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINPX_2V4

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINPX_2V4 (void)

Function description

Return the calibration value for the gain of single ended positive input at range 2.4 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINPX_2V4

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINPX_2V4 (void)

Function description

Return the calibration value for the offset of single ended positive input at range 2.4 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINPX_1V2

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINPX_1V2 (void)

Function description

Return the calibration value for the gain of single ended positive input at range 1.2 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINPX_1V2

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINPX_1V2 (void)

Function description

Return the calibration value for the offset of single ended positive input at range 1.2 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINMX_3V6

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINMX_3V6 (void)

Function description

Return the calibration value for the gain of single ended negative input at range 3.6 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINMX_3V6

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINMX_3V6 (void)

Function description

Return the calibration value for the offset of single ended negative input at range 3.6 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINMX_2V4

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINMX_2V4 (void)

Function description

Return the calibration value for the gain of single ended negative input at range 2.4 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINMX_2V4

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINMX_2V4 (void)

Function description

Return the calibration value for the offset of single ended negative input at range 2.4 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINMX_1V2

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINMX_1V2 (void)

Function description

Return the calibration value for the gain of single ended negative input at range 1.2 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINMX_1V2

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINMX_1V2 (void)

Function description

Return the calibration value for the offset of single ended negative input at range 1.2 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINDIFF_3V6

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINDIFF_3V6 (void)

Function description

Return the calibration value for the gain of differential input at range 3.6 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINDIFF_3V6

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINDIFF_3V6 (void)

Function description

Return the calibration value for the offset of differential input at range 3.6 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINDIFF_2V4

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINDIFF_2V4 (void)

Function description

Return the calibration value for the gain of differential input at range 2.4 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINDIFF_2V4

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINDIFF_2V4 (void)

Function description

Return the calibration value for the offset of differential input at range 2.4 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_GAIN_FOR_VINDIFF_1V2

Function name

__STATIC_INLINE uint16_t LL_ADC_GET_CALIB_GAIN_FOR_VINDIFF_1V2 (void)

Function description

Return the calibration value for the gain of differential input at range 1.2 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

LL_ADC_GET_CALIB_OFFSET_FOR_VINDIFF_1V2

Function name

__STATIC_INLINE int8_t LL_ADC_GET_CALIB_OFFSET_FOR_VINDIFF_1V2 (void)

Function description

Return the calibration value for the offset of differential input at range 1.2 V.

Return values

- **Returned:** value to be put in COMP_X register if 0xFFFFFFFF means no calibration point.

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

- 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.

LL_ADC_Init

Function name

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

Function description

Initialize some features of ADC instance.

Parameters

- **ADCx:** ADC instance
- **pADC_InitStruct:** Pointer to a LL_ADC_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 ADC Configuration functions.
- 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.

LL_ADC_VoltageRangeInit

Function name

ErrorStatus LL_ADC_VoltageRangeInit (ADC_TypeDef * ADCx, const LL_ADC_VoltRangeInitTypeDef * pVoltRange_InitStruct)

Function description

Configure the input voltage range for all the inputs.

Parameters

- **ADCx:** ADC instance
- **pVoltRange_InitStruct:** pointer to a LL_ADC_VoltRangeInitTypeDef structure that contains the voltage range for all the inputs.

Return values

- **None:**

LL_ADC_SequenceInit

Function name

ErrorStatus LL_ADC_SequenceInit (ADC_TypeDef * ADCx, const LL_ADC_SequenceInitTypeDef * pSequence_InitStruct)

Function description

Configure the channels for the conversion sequence.

Parameters

- **ADCx:** ADC instance
- **pSequence_InitStruct:** pointer to a LL_ADC_SequenceInitTypeDef structure that contains the channel codes for the conversion sequence.

Return values

- **ErrorStatus:**

44.3 ADC Firmware driver defines

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

44.3.1 ADC

ADC

ADC watchdog channel selection for bit mask definitions

LL_ADC_AWD_CH_VINM0

ADC watchdog channel selection: VINM0 to negative input

LL_ADC_AWD_CH_VINM1

ADC watchdog channel selection: VINM1 to negative input

LL_ADC_AWD_CH_VINM2

ADC watchdog channel selection: VINM2 to negative input

LL_ADC_AWD_CH_VINM3

ADC watchdog channel selection: VINM3 to negative input

LL_ADC_AWD_CH_VINP0

ADC watchdog channel selection: VINP0 to positive input

LL_ADC_AWD_CH_VINP1

ADC watchdog channel selection: VINP1 to positive input

LL_ADC_AWD_CH_VINP2

ADC watchdog channel selection: VINP2 to positive input

LL_ADC_AWD_CH_VINP3

ADC watchdog channel selection: VINP3 to positive input

LL_ADC_AWD_CH_MICROM

ADC watchdog channel selection: MICROM to negative input

LL_ADC_AWD_CH_MICROP

ADC watchdog channel selection: MICROP to positive input

LL_ADC_AWD_CH_VBAT

ADC watchdog channel selection: VBAT to negative input

LL_ADC_AWD_CH_TEMPSENSOR

ADC watchdog channel selection: TEMP to positive input

LL_ADC_AWD_CH_VDDA_NEG

ADC watchdog channel selection: VDDA to negative input

LL_ADC_AWD_CH_VDDA_POS

ADC watchdog channel selection: VDDA to positive input

LL_ADC_AWD_CH_GND_NEG

ADC watchdog channel selection: GND to negative input

LL_ADC_AWD_CH_GND_POS

ADC watchdog channel selection: GND to positive input

ADC calibration points location

ADC_LAYOUT_ID

ADC_CALIB_ADDRESS_VINPX_3V6

ADC_CALIB_ADDRESS_VINMX_3V6

ADC_CALIB_ADDRESS_VINDIFF_3V6

ADC_CALIB_ADDRESS_VINPX_2V4

ADC_CALIB_ADDRESS_VINMX_2V4

ADC_CALIB_ADDRESS_VINDIFF_2V4

ADC_CALIB_ADDRESS_VINPX_1V2

ADC_CALIB_ADDRESS_VINMX_1V2

ADC_CALIB_ADDRESS_VINDIFF_1V2

ADC calibration points definitions

LL_ADC_CALIB_POINT_1

ADC calibration point 1

LL_ADC_CALIB_POINT_2

ADC calibration point 2

LL_ADC_CALIB_POINT_3

ADC calibration point 3

LL_ADC_CALIB_POINT_4

ADC calibration point 4

ADC channel number code for conversion definitions

LL_ADC_CHANNEL_VINM0

ADC channel for VINM0 to single negative input

LL_ADC_CHANNEL_VINM1

ADC channel for VINM1 to single negative input

LL_ADC_CHANNEL_VINM2

ADC channel for VINM2 to single negative input

LL_ADC_CHANNEL_VINM3

ADC channel for VINM3 to single negative input

LL_ADC_CHANNEL_VINP0

ADC channel for VINP0 to single positive input

LL_ADC_CHANNEL_VINP1

ADC channel for VINP1 to single positive input

LL_ADC_CHANNEL_VINP2

ADC channel for VINP2 to single positive input

LL_ADC_CHANNEL_VINP3

ADC channel for VINP3 to single positive input

LL_ADC_CHANNEL_VINP0_VINM0

ADC channel for VINP0 - VINM0 to differential input

LL_ADC_CHANNEL_VINP1_VINM1

ADC channel for VINP1 - VINM1 to differential input

LL_ADC_CHANNEL_VINP2_VINM2

ADC channel for VINP2 - VINM2 to differential input

LL_ADC_CHANNEL_VINP3_VINM3

ADC channel for VINP3 - VINM3 to differential input

LL_ADC_CHANNEL_VBAT

ADC channel for VBAT, battery level detector

LL_ADC_CHANNEL_TEMPSENSOR

ADC channel for temperature sensor

ADC default range value definitions

LL_ADC_DEFAULT_RANGE_VALUE_1V2

ADC default range value up to 1.2 V

LL_ADC_DEFAULT_RANGE_VALUE_2V4

ADC default range value up to 2.4 V

LL_ADC_DEFAULT_RANGE_VALUE_3V6

ADC default range value up to 3.6 V

ADC Down Sampler data width definitions

LL_ADC_DS_DATA_WIDTH_12_BIT

ADC Down Sampler data width 12 bits

LL_ADC_DS_DATA_WIDTH_13_BIT

ADC Down Sampler data width 13 bits

LL_ADC_DS_DATA_WIDTH_14_BIT

ADC Down Sampler data width 14 bits

LL_ADC_DS_DATA_WIDTH_15_BIT

ADC Down Sampler data width 15 bits

LL_ADC_DS_DATA_WIDTH_16_BIT

ADC Down Sampler data width 16 bits

ADC Down Sampler ratio definitions

LL_ADC_DS_RATIO_1

ADC Down Sampler ratio 1, no down sampling (default)

LL_ADC_DS_RATIO_2

ADC Down Sampler ratio 2

LL_ADC_DS_RATIO_4

ADC Down Sampler ratio 4

LL_ADC_DS_RATIO_8

ADC Down Sampler ratio 8

LL_ADC_DS_RATIO_16

ADC Down Sampler ratio 16

LL_ADC_DS_RATIO_32

ADC Down Sampler ratio 32

LL_ADC_DS_RATIO_64

ADC Down Sampler ratio 64

LL_ADC_DS_RATIO_128

ADC Down Sampler ratio 128

ADC - Sequencer ranks

LL_ADC_RANK_1

ADC rank 1

LL_ADC_RANK_2

ADC rank 2

LL_ADC_RANK_3

ADC rank 3

LL_ADC_RANK_4

ADC rank 4

LL_ADC_RANK_5

ADC rank 5

LL_ADC_RANK_6

ADC rank 6

LL_ADC_RANK_7

ADC rank 7

LL_ADC_RANK_8

ADC rank 8

LL_ADC_RANK_9

ADC rank 9

LL_ADC_RANK_10

ADC rank 10

LL_ADC_RANK_11

ADC rank 11

LL_ADC_RANK_12

ADC rank 12

LL_ADC_RANK_13

ADC rank 13

LL_ADC_RANK_14

ADC rank 14

LL_ADC_RANK_15

ADC rank 15

LL_ADC_RANK_16

ADC rank 16

ADC helper macro

LL_ADC_TEMPERATURE_CALC_ERROR

__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_WIDTH__: This parameter can be one of the following values:
  – LL_ADC_DS_DATA_WIDTH_12_BIT
  – LL_ADC_DS_DATA_WIDTH_13_BIT
  – LL_ADC_DS_DATA_WIDTH_14_BIT
  – LL_ADC_DS_DATA_WIDTH_15_BIT
  – LL_ADC_DS_DATA_WIDTH_16_BIT

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_REFERENCE_VOLTAGE

Description:

- Helper macro to define the ADC conversion data full-scale analog voltage value corresponding to the selected ADC input range.

Parameters:

- __INPUT_VOLTAGE_RANGE__: This parameter can be one of the following values:
  – LL_ADC_VIN_RANGE_3V6
  – LL_ADC_VIN_RANGE_2V4
  – LL_ADC_VIN_RANGE_1V2

Return value:

- ADC: conversion data full-scale digital value (unit: digital value of ADC conversion data)

__LL_ADC_IS_CHANNEL_INTERNAL

Description:

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

Parameters:

- __CHANNEL__: This parameter can be one of the following values:
  - LL_ADC_CHANNEL_VINM0
  - LL_ADC_CHANNEL_VINM1
  - LL_ADC_CHANNEL_VINM2
  - LL_ADC_CHANNEL_VINM3
  - LL_ADC_CHANNEL_VINP0
  - LL_ADC_CHANNEL_VINP1
  - LL_ADC_CHANNEL_VINP2
  - LL_ADC_CHANNEL_VINP3
  - LL_ADC_CHANNEL_VINP0_VINM0
  - LL_ADC_CHANNEL_VINP1_VINM1
  - LL_ADC_CHANNEL_VINP2_VINM2
  - LL_ADC_CHANNEL_VINP3_VINM3
  - LL_ADC_CHANNEL_VBAT
  - LL_ADC_CHANNEL_TEMPSENSOR

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.

__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_WIDTH_CURRENT__: Resolution of the data to be converted This parameter can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT
- __ADC_WIDTH_TARGET__: Resolution of the data after conversion This parameter can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT

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:

- __INPUT_VOLTAGE_RANGE__: Analog reference voltage (unit: mV)
  - LL_ADC_VIN_RANGE_3V6
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_1V2
- __ADC_DATA__: ADC conversion data (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT

Return value:

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

__LL_ADC_CALC_DIFF_DATA_TO_VOLTAGE

Description:

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

Parameters:

- __INPUT_VOLTAGE_RANGE__: Analog reference voltage (unit: mV)
  - LL_ADC_VIN_RANGE_3V6
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_1V2
- __ADC_DATA__: ADC conversion data (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT

Return value:

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

Notes:

- ADC data from ADC data register is unsigned and centered around middle code in. Converted voltage can be positive or negative depending on differential input voltages.

__LL_ADC_CALC_TEMPERATURE

Description:

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

Parameters:

- __TEMPSENSOR_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT

Return value:

- Temperature: (unit: degree Celsius) In case or error, value LL_ADC_TEMPERATURE_CALC_ERROR is returned (inconsistent temperature value)

Notes:

- Computation is using temperature sensor calibration values stored in system memory for each device during production. Calculation formula: Temperature = (55 / (C85 - C30)) * (TS_ADC_DATA - C30) + 30 with TS_ADC_DATA = temperature sensor raw data measured by ADC C85 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL1 (calibrated in factory) C30 = equivalent TS_ADC_DATA at temperature TEMP_DEGC_CAL2 (calibrated in factory) Caution: Calculation relevancy under reserve that calibration parameters are correct (address and data). To calculate temperature using temperature sensor datasheet typical values (generic values less, therefore less accurate than calibrated values), use helper macro __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS().

__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_ADC_DATA__: ADC conversion data of internal temperature sensor (unit: digital value).
- __ADC_WIDTH__: This parameter can be one of the following values:
  - LL_ADC_DS_DATA_WIDTH_12_BIT
  - LL_ADC_DS_DATA_WIDTH_13_BIT
  - LL_ADC_DS_DATA_WIDTH_14_BIT
  - LL_ADC_DS_DATA_WIDTH_15_BIT
  - LL_ADC_DS_DATA_WIDTH_16_BIT

Return value:

- Temperature: (unit: degree Celsius)

Notes:

- Computation is using temperature sensor typical values (refer to device datasheet). Calculation formula: Temperature = (Avg_Slope * (TS_ADC_DATA - VMIN) + TEMP_MIN

LL_ADC_CONVERT_VOLTAGE_TO_DATA_AWD_THRESHOLD_SINGLE

Description:

- Helper macro to convert the threshold value from mV to code for single conversion or battery sensor.

Parameters:

- __VOLTAGE_MV__: The voltage threshold for the ADC watchdog in mV.
- __INPUT_VOLTAGE_RANGE__: Analog reference voltage (unit: mV)
  - LL_ADC_VIN_RANGE_3V6
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_1V2

Return value:

- Returned: 12-bit value to be used as ADC watchdog threshold.

LL_ADC_CONVERT_VOLTAGE_TO_DATA_AWD_THRESHOLD_DIFF

Description:

- Helper macro to convert the threshold value from mV to code for differential conversion.

Parameters:

- __VOLTAGE_MV__: The voltage threshold for the ADC watchdog in mV.
- __INPUT_VOLTAGE_RANGE__: Analog reference voltage (unit: mV)
  - LL_ADC_VIN_RANGE_3V6
  - LL_ADC_VIN_RANGE_2V4
  - LL_ADC_VIN_RANGE_1V2

Return value:

- Returned: 12-bit value to be used as ADC watchdog threshold.

Common write and read registers Macros

__ADC_PTR_REG_OFFSET

Description:

- Driver macro reserved for internal use: set a pointer to a register from a register basis from which an offset is applied.

Parameters:

- __REG__: Register basis from which the offset is applied.
- __REG_OFFFSET__: Offset to be applied (unit: number of registers).

Return value:

- Pointer: to register address

ADC input voltage range definitions

LL_ADC_VIN_RANGE_1V2

ADC input voltage range up to 1.2 V

LL_ADC_VIN_RANGE_2V4

ADC input voltage range up to 2.4 V

LL_ADC_VIN_RANGE_3V6

ADC input voltage range up to 3.6 V

ADC interrupts enable definitions

LL_ADC_IRQ_EN_OVRDS

ADC IRQ enable OVRDS

LL_ADC_IRQ_EN_AWD1

ADC IRQ enable AWD1

LL_ADC_IRQ_EN_EOC

ADC IRQ enable EOC

LL_ADC_IRQ_EN_EOS

ADC IRQ enable EOS

LL_ADC_IRQ_EN_EODS

ADC IRQ enable EODS

ADC IRQ_STATUS register mask definitions

LL_ADC_IRQ_FLAG_OVRDS

ADC IRQ flag OVRDS

LL_ADC_IRQ_FLAG_AWD1

ADC IRQ flag AWD1

LL_ADC_IRQ_FLAG_EOC

ADC IRQ flag EOC

LL_ADC_IRQ_FLAG_EOS

ADC IRQ flag EOS

LL_ADC_IRQ_FLAG_EODS

ADC IRQ flag EODS

LL_ADC_IRQ_FLAGS_MASK

ADC overrun configuration definitions

LL_ADC_NEW_DATA_IS_LOST

Previous data is preserved, new data is lost.

LL_ADC_NEW_DATA_IS_KEPT

Previous data is overwritten, new data is kept.

ADC sample rate definitions

LL_ADC_SAMPLE_RATE_16

ADC conversion rate at F_ADC_CLK / 16

LL_ADC_SAMPLE_RATE_20

ADC conversion rate at F_ADC_CLK / 20

LL_ADC_SAMPLE_RATE_24

ADC conversion rate at F_ADC_CLK / 24

LL_ADC_SAMPLE_RATE_28

ADC conversion rate at F_ADC_CLK / 28

LL_ADC_SAMPLE_RATE_32

ADC conversion rate at F_ADC_CLK / 32

LL_ADC_SAMPLE_RATE_36

ADC conversion rate at F_ADC_CLK / 36

LL_ADC_SAMPLE_RATE_40
ADC conversion rate at F_ADC_CLK / 40

LL_ADC_SAMPLE_RATE_44
ADC conversion rate at F_ADC_CLK / 44

LL_ADC_SAMPLE_RATE_48
ADC conversion rate at F_ADC_CLK / 48

LL_ADC_SAMPLE_RATE_52
ADC conversion rate at F_ADC_CLK / 52

LL_ADC_SAMPLE_RATE_56
ADC conversion rate at F_ADC_CLK / 56

LL_ADC_SAMPLE_RATE_60
ADC conversion rate at F_ADC_CLK / 60

LL_ADC_SAMPLE_RATE_64
ADC conversion rate at F_ADC_CLK / 64

LL_ADC_SAMPLE_RATE_68
ADC conversion rate at F_ADC_CLK / 68

LL_ADC_SAMPLE_RATE_72
ADC conversion rate at F_ADC_CLK / 72

LL_ADC_SAMPLE_RATE_76
ADC conversion rate at F_ADC_CLK / 76

LL_ADC_SAMPLE_RATE_80
ADC conversion rate at F_ADC_CLK / 80

LL_ADC_SAMPLE_RATE_84
ADC conversion rate at F_ADC_CLK / 84

LL_ADC_SAMPLE_RATE_88
ADC conversion rate at F_ADC_CLK / 88

LL_ADC_SAMPLE_RATE_92
ADC conversion rate at F_ADC_CLK / 92

LL_ADC_SAMPLE_RATE_96
ADC conversion rate at F_ADC_CLK / 96

LL_ADC_SAMPLE_RATE_100
ADC conversion rate at F_ADC_CLK / 100

LL_ADC_SAMPLE_RATE_104
ADC conversion rate at F_ADC_CLK / 104

LL_ADC_SAMPLE_RATE_108
ADC conversion rate at F_ADC_CLK / 108

LL_ADC_SAMPLE_RATE_112
ADC conversion rate at F_ADC_CLK / 112

LL_ADC_SAMPLE_RATE_116

ADC conversion rate at F_ADC_CLK / 116

LL_ADC_SAMPLE_RATE_120

ADC conversion rate at F_ADC_CLK / 120

LL_ADC_SAMPLE_RATE_124

ADC conversion rate at F_ADC_CLK / 124

LL_ADC_SAMPLE_RATE_128

ADC conversion rate at F_ADC_CLK / 128

LL_ADC_SAMPLE_RATE_132

ADC conversion rate at F_ADC_CLK / 132

LL_ADC_SAMPLE_RATE_136

ADC conversion rate at F_ADC_CLK / 136

LL_ADC_SAMPLE_RATE_140

ADC conversion rate at F_ADC_CLK / 140

ADC input sampling method definitions

LL_ADC_SAMPLING_AT_START

Sampling only at conversion start

LL_ADC_SAMPLING_AT_END

Sampling starts at the end of conversion (default)

45 LL BUS Generic Driver

45.1 BUS Firmware driver API description

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

45.1.1 Detailed description of functions

LL_AHB1_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable AHB peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA
  - LL_AHB1_GRP1_PERIPH_GPIOA
  - LL_AHB1_GRP1_PERIPH_GPIOB
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_PKA
  - LL_AHB1_GRP1_PERIPH_RNG

Return values

- **None:**

Notes

- LL_AHB1_GRP1_PERIPH_PKA

Reference Manual to LL API cross reference:

- AHBENR DMAEN LL_AHB1_GRP1_EnableClock AHBENR GPIOAEN LL_AHB1_GRP1_EnableClock AHBENR GPIOBEN LL_AHB1_GRP1_EnableClock AHBENR CRCEN LL_AHB1_GRP1_EnableClock AHBENR PKAEN LL_AHB1_GRP1_EnableClock AHBENR RNGEN 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 AHB peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_AHB1_GRP1_PERIPH_DMA
  - LL_AHB1_GRP1_PERIPH_GPIOA
  - LL_AHB1_GRP1_PERIPH_GPIOB
  - LL_AHB1_GRP1_PERIPH_CRC
  - LL_AHB1_GRP1_PERIPH_PKA
  - LL_AHB1_GRP1_PERIPH_RNG

Return values

- **uint32_t:**

Notes

- LL_AHB1_GRP1_PERIPH_PKA

Reference Manual to LL API cross reference:

- AHBENR DMAEN LL_AHB1_GRP1_IsEnabledClock AHBENR GPIOAEN LL_AHB1_GRP1_IsEnabledClock AHBENR GPIOBEN LL_AHB1_GRP1_IsEnabledClock AHBENR CRCEN LL_AHB1_GRP1_IsEnabledClock AHBENR PAKAEN LL_AHB1_GRP1_IsEnabledClock AHBENR RNGEN LL_AHB1_GRP1_IsEnabledClock

LL_AHB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_AHB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable AHB peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_DMA
  – LL_AHB1_GRP1_PERIPH_GPIOA
  – LL_AHB1_GRP1_PERIPH_GPIOB
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_PKA
  – LL_AHB1_GRP1_PERIPH_RNG

Return values

- **None:**

Notes

- LL_AHB1_GRP1_PERIPH_PKA

Reference Manual to LL API cross reference:

- AHBENR DMAENN LL_AHB1_GRP1_DisableClock AHBENR GPIOAEN LL_AHB1_GRP1_DisableClock AHBENR GPIOBEN LL_AHB1_GRP1_DisableClock AHBENR CRCEN LL_AHB1_GRP1_DisableClock AHBENR PKAEN LL_AHB1_GRP1_DisableClock AHBENR RNGEN LL_AHB1_GRP1_DisableClock

LL_AHB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force AHB peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_DMA
  – LL_AHB1_GRP1_PERIPH_GPIOA
  – LL_AHB1_GRP1_PERIPH_GPIOB
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_PKA
  – LL_AHB1_GRP1_PERIPH_RNG

Return values

- **None:**

Notes

- LL_AHB1_GRP1_PERIPH_PKA

Reference Manual to LL API cross reference:

- AHBRSTR DMARST LL_AHB1_GRP1_ForceReset AHBRSTR GPIOARST LL_AHB1_GRP1_ForceReset AHBRSTR GPIOBRST LL_AHB1_GRP1_ForceReset AHBRSTR CRCRST LL_AHB1_GRP1_ForceReset AHBRSTR PKARST LL_AHB1_GRP1_ForceReset AHBRSTR RNGRST LL_AHB1_GRP1_ForceReset

LL_AHB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_AHB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release AHB peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_AHB1_GRP1_PERIPH_DMA
  – LL_AHB1_GRP1_PERIPH_GPIOA
  – LL_AHB1_GRP1_PERIPH_GPIOB
  – LL_AHB1_GRP1_PERIPH_CRC
  – LL_AHB1_GRP1_PERIPH_PKA
  – LL_AHB1_GRP1_PERIPH_RNG

Return values

- **None:**

Notes

- LL_AHB1_GRP1_PERIPH_PKA

Reference Manual to LL API cross reference:

- AHBRSTR DMARST LL_AHB1_GRP1_ReleaseReset AHBRSTR GPIOARST LL_AHB1_GRP1_ReleaseReset AHBRSTR GPIOBRST LL_AHB1_GRP1_ReleaseReset AHBRSTR CRCRST LL_AHB1_GRP1_ReleaseReset AHBRSTR PKARST LL_AHB1_GRP1_ReleaseReset AHBRSTR RNGRST LL_AHB1_GRP1_ReleaseReset

LL_APB0_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB0_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB0 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB0_GRP1_PERIPH_TIM1
  – LL_APB0_GRP1_PERIPH_TIM2
  – LL_APB0_GRP1_PERIPH_TIM16
  – LL_APB0_GRP1_PERIPH_TIM17
  – LL_APB0_GRP1_PERIPH_SYSCFG
  – LL_APB0_GRP1_PERIPH_RTC
  – LL_APB0_GRP1_PERIPH_WDG

Return values

- **None:**

Notes

- LL_APB0_GRP1_PERIPH_TIM1 is valid only for STM32WB06 and STM32WB07
- LL_APB0_GRP1_PERIPH_TIM2, LL_APB0_GRP1_PERIPH_TIM16 and LL_APB0_GRP1_PERIPH_TIM17 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- APB0ENR TIM1EN LL_APB0_GRP1_EnableClock APB0ENR TIM2EN LL_APB0_GRP1_EnableClock APB0ENR TIM16EN LL_APB0_GRP1_EnableClock APB0ENR TIM17EN LL_APB0_GRP1_EnableClock APB0ENR SYSCFGEN LL_APB0_GRP1_EnableClock APB0ENR RTCEN LL_APB0_GRP1_EnableClock APB0ENR WDGEN LL_APB0_GRP1_EnableClock

LL_APB0_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB0_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB0 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB0_GRP1_PERIPH_TIM1
  – LL_APB0_GRP1_PERIPH_TIM2
  – LL_APB0_GRP1_PERIPH_TIM16
  – LL_APB0_GRP1_PERIPH_TIM17
  – LL_APB0_GRP1_PERIPH_SYSCFG
  – LL_APB0_GRP1_PERIPH_RTC
  – LL_APB0_GRP1_PERIPH_WDG

Return values

- **uint32_t:**

Notes

- LL_APB0_GRP1_PERIPH_TIM1 is valid only for STM32WB06 and STM32WB07
- LL_APB0_GRP1_PERIPH_TIM2, LL_APB0_GRP1_PERIPH_TIM16 and LL_APB0_GRP1_PERIPH_TIM17 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- APB0ENR TIM1EN LL_APB0_GRP1_IsEnabledClock APB0ENR TIM2EN LL_APB0_GRP1_IsEnabledClock APB0ENR TIM16EN LL_APB0_GRP1_IsEnabledClock APB0ENR TIM17EN LL_APB0_GRP1_IsEnabledClock APB0ENR SYSCFGEN LL_APB0_GRP1_IsEnabledClock APB0ENR RTCEN LL_APB0_GRP1_IsEnabledClock APB0ENR WDGEN LL_APB0_GRP1_IsEnabledClock

LL_APB0_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB0_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB0 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB0_GRP1_PERIPH_TIM1
  – LL_APB0_GRP1_PERIPH_TIM2
  – LL_APB0_GRP1_PERIPH_TIM16
  – LL_APB0_GRP1_PERIPH_TIM17
  – LL_APB0_GRP1_PERIPH_SYSCFG
  – LL_APB0_GRP1_PERIPH_RTC
  – LL_APB0_GRP1_PERIPH_WDG

Return values

- **None:**

Notes

- LL_APB0_GRP1_PERIPH_TIM1 is valid only for STM32WB06 and STM32WB07
- LL_APB0_GRP1_PERIPH_TIM2, LL_APB0_GRP1_PERIPH_TIM16 and LL_APB0_GRP1_PERIPH_TIM17 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- APB0ENR TIM1EN LL_APB0_GRP1_DisableClock APB0ENR TIM2EN LL_APB0_GRP1_DisableClock APB0ENR TIM16EN LL_APB0_GRP1_DisableClock APB0ENR TIM17EN LL_APB0_GRP1_DisableClock APB0ENR SYSCFGEN LL_APB0_GRP1_DisableClock APB0ENR RTCEN LL_APB0_GRP1_DisableClock APB0ENR WDGEN LL_APB0_GRP1_DisableClock

LL_APB0_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB0_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB0 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB0_GRP1_PERIPH_TIM1
  – LL_APB0_GRP1_PERIPH_TIM2
  – LL_APB0_GRP1_PERIPH_TIM16
  – LL_APB0_GRP1_PERIPH_TIM17
  – LL_APB0_GRP1_PERIPH_SYSCFG
  – LL_APB0_GRP1_PERIPH_RTC
  – LL_APB0_GRP1_PERIPH_WDG

Return values

- **None:**

Notes

- LL_APB0_GRP1_PERIPH_TIM1 is valid only for STM32WB06 and STM32WB07
- LL_APB0_GRP1_PERIPH_TIM2, LL_APB0_GRP1_PERIPH_TIM16 and LL_APB0_GRP1_PERIPH_TIM17 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- APB0RSTR TIM1RST LL_APB0_GRP1_ForceReset APB0RSTR TIM2RST LL_APB0_GRP1_ForceReset APB0RSTR TIM16RST LL_APB0_GRP1_ForceReset APB0RSTR TIM17RST LL_APB0_GRP1_ForceReset APB0RSTR SYSCFGRST LL_APB0_GRP1_ForceReset APB0RSTR DACRST LL_APB0_GRP1_ForceReset APB0RSTR RTCRST LL_APB0_GRP1_ForceReset APB0RSTR WDGRST LL_APB0_GRP1_ForceReset

LL_APB0_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB0_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB0 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB0_GRP1_PERIPH_TIM1
  - LL_APB0_GRP1_PERIPH_TIM2
  - LL_APB0_GRP1_PERIPH_TIM16
  - LL_APB0_GRP1_PERIPH_TIM17
  - LL_APB0_GRP1_PERIPH_SYSCFG
  - LL_APB0_GRP1_PERIPH_RTC
  - LL_APB0_GRP1_PERIPH_WDG

Return values

- **None:**

Notes

- LL_APB0_GRP1_PERIPH_TIM1 is valid only for STM32WB06 and STM32WB07
- LL_APB0_GRP1_PERIPH_TIM2, LL_APB0_GRP1_PERIPH_TIM16 and LL_APB0_GRP1_PERIPH_TIM17 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- APB0RSTR TIM1RST LL_APB0_GRP1_ReleaseReset APB0RSTR TIM2RST LL_APB0_GRP1_ReleaseReset APB0RSTR TIM16RST LL_APB0_GRP1_ReleaseReset APB0RSTR TIM17RST LL_APB0_GRP1_ReleaseReset APB0RSTR SYSCFGRST LL_APB0_GRP1_ReleaseReset APB0RSTR DACRST LL_APB0_GRP1_ReleaseReset APB0RSTR RTCRST LL_APB0_GRP1_ReleaseReset APB0RSTR WDGRST LL_APB0_GRP1_ReleaseReset

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_SPI1
  - LL_APB1_GRP1_PERIPH_ADCDIG
  - LL_APB1_GRP1_PERIPH_ADCANA
  - LL_APB1_GRP1_PERIPH_LPUART1
  - LL_APB1_GRP1_PERIPH_USART1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2

Return values

- **None:**

Notes

- LL_APB1_GRP1_PERIPH_SPI1, LL_APB1_GRP1_PERIPH_SPI2 and LL_APB1_GRP1_PERIPH_I2C2 are valid for STM32WB06 and STM32WB07

Reference Manual to LL API cross reference:

- APB1ENR SPI1EN LL_APB1_GRP1_EnableClock APB1ENR ADCDIGEN LL_APB1_GRP1_EnableClock APB1ENR ADCANAEN LL_APB1_GRP1_EnableClock APB1ENR LPUARTEN LL_APB1_GRP1_EnableClock APB1ENR USARTEN LL_APB1_GRP1_EnableClock APB1ENR SPI2EN LL_APB1_GRP1_EnableClock APB1ENR SPI3EN LL_APB1_GRP1_EnableClock APB1ENR I2C1EN LL_APB1_GRP1_EnableClock APB1ENR I2C2EN 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 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_SPI1
  - LL_APB1_GRP1_PERIPH_ADCDIG
  - LL_APB1_GRP1_PERIPH_ADCANA
  - LL_APB1_GRP1_PERIPH_LPUART1
  - LL_APB1_GRP1_PERIPH_USART1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2

Return values

- **uint32_t:**

Notes

- LL_APB1_GRP1_PERIPH_SPI1, LL_APB1_GRP1_PERIPH_SPI2 and LL_APB1_GRP1_PERIPH_I2C2 are valid for STM32WB06 and STM32WB07

Reference Manual to LL API cross reference:

- APB1ENR SPI1EN LL_APB1_GRP1_IsEnabledClock APB1ENR ADCDIGEN LL_APB1_GRP1_IsEnabledClock APB1ENR ADCANAEN LL_APB1_GRP1_IsEnabledClock APB1ENR LPUARTEN LL_APB1_GRP1_IsEnabledClock APB1ENR USARTEN LL_APB1_GRP1_IsEnabledClock APB1ENR SPI2EN LL_APB1_GRP1_IsEnabledClock APB1ENR SPI3EN LL_APB1_GRP1_IsEnabledClock APB1ENR I2C1EN LL_APB1_GRP1_IsEnabledClock APB1ENR I2C2EN LL_APB1_GRP1_IsEnabledClock

LL_APB1_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB1_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB1 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_SPI1
  - LL_APB1_GRP1_PERIPH_ADCDIG
  - LL_APB1_GRP1_PERIPH_ADCANA
  - LL_APB1_GRP1_PERIPH_LPUART1
  - LL_APB1_GRP1_PERIPH_USART1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2

Return values

- **None:**

Notes

- LL_APB1_GRP1_PERIPH_SPI1, LL_APB1_GRP1_PERIPH_SPI2 and LL_APB1_GRP1_PERIPH_I2C2 are valid for STM32WB06 and STM32WB07

Reference Manual to LL API cross reference:

- APB1ENR SPI1EN LL_APB1_GRP1_IsEnabledClock APB1ENR ADCDIGEN LL_APB1_GRP1_DisableClock APB1ENR ADCANAEN LL_APB1_GRP1_DisableClock APB1ENR LPUARTEN LL_APB1_GRP1_DisableClock APB1ENR USARTEN LL_APB1_GRP1_DisableClock APB1ENR SPI2EN LL_APB1_GRP1_DisableClock APB1ENR SPI3EN LL_APB1_GRP1_DisableClock APB1ENR I2C1EN LL_APB1_GRP1_DisableClock APB1ENR I2C2EN LL_APB1_GRP1_DisableClock

LL_APB1_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  - LL_APB1_GRP1_PERIPH_ALL
  - LL_APB1_GRP1_PERIPH_SPI1
  - LL_APB1_GRP1_PERIPH_ADCDIG
  - LL_APB1_GRP1_PERIPH_LPUART1
  - LL_APB1_GRP1_PERIPH_USART1
  - LL_APB1_GRP1_PERIPH_SPI2
  - LL_APB1_GRP1_PERIPH_SPI3
  - LL_APB1_GRP1_PERIPH_I2C1
  - LL_APB1_GRP1_PERIPH_I2C2

Return values

- **None:**

Notes

- LL_APB1_GRP1_PERIPH_SPI1, LL_APB1_GRP1_PERIPH_SPI2 and LL_APB1_GRP1_PERIPH_I2C2 are valid for STM32WB06 and STM32WB07

Reference Manual to LL API cross reference:

- APB1RSTR SPI1RST LL_APB1_GRP1_ForceReset APB1RSTR ADCRST LL_APB1_GRP1_ForceReset APB1RSTR LPUARTRST LL_APB1_GRP1_ForceReset APB1RSTR USARTRST LL_APB1_GRP1_ForceReset APB1RSTR SPI2RST LL_APB1_GRP1_ForceReset APB1RSTR SPI3RST LL_APB1_GRP1_ForceReset APB1RSTR I2C1RST LL_APB1_GRP1_ForceReset APB1RSTR I2C2RST LL_APB1_GRP1_ForceReset

LL_APB1_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB1_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB1 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB1_GRP1_PERIPH_ALL
  – LL_APB1_GRP1_PERIPH_SPI1
  – LL_APB1_GRP1_PERIPH_ADCDIG
  – LL_APB1_GRP1_PERIPH_LPUART1
  – LL_APB1_GRP1_PERIPH_USART1
  – LL_APB1_GRP1_PERIPH_SPI2
  – LL_APB1_GRP1_PERIPH_SPI3
  – LL_APB1_GRP1_PERIPH_I2C1
  – LL_APB1_GRP1_PERIPH_I2C2

Return values

- **None:**

Notes

- LL_APB1_GRP1_PERIPH_SPI1, LL_APB1_GRP1_PERIPH_SPI2 and LL_APB1_GRP1_PERIPH_I2C2 are valid for STM32WB06 and STM32WB07

Reference Manual to LL API cross reference:

- APB1RSTR SPI1RST LL_APB1_GRP1_ReleaseReset APB1RSTR ADCRST LL_APB1_GRP1_ReleaseReset APB1RSTR LPUARTRST LL_APB1_GRP1_ReleaseReset APB1RSTR USARTRST LL_APB1_GRP1_ReleaseReset APB1RSTR SPI2RST LL_APB1_GRP1_ReleaseReset APB1RSTR SPI3RST LL_APB1_GRP1_ReleaseReset APB1RSTR I2C1RST LL_APB1_GRP1_ReleaseReset APB1RSTR I2C2RST LL_APB1_GRP1_ReleaseReset

LL_APB2_GRP1_EnableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_EnableClock (uint32_t Periphs)

Function description

Enable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_MRBLE

Return values

- **None:**

Notes

- LL_APB2_GRP1_PERIPH_MRBLE

Reference Manual to LL API cross reference:

- APB2ENR MRBLEEN LL_APB2_GRP1_EnableClock

LL_APB2_GRP1_IsEnabledClock

Function name

__STATIC_INLINE uint32_t LL_APB2_GRP1_IsEnabledClock (uint32_t Periphs)

Function description

Check if APB2 peripheral clock is enabled or not.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_MRBLE

Return values

- **uint32_t:**

Notes

- LL_APB2_GRP1_PERIPH_MRBLE

Reference Manual to LL API cross reference:

- APB2ENR MRBLEEN LL_APB2_GRP1_IsEnabledClock

LL_APB2_GRP1_DisableClock

Function name

__STATIC_INLINE void LL_APB2_GRP1_DisableClock (uint32_t Periphs)

Function description

Disable APB2 peripherals clock.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_MRBLE

Return values

- **None:**

Notes

- LL_APB2_GRP1_PERIPH_MRBLE

Reference Manual to LL API cross reference:

- APB2ENR MRBLEEN LL_APB2_GRP1_DisableClock

LL_APB2_GRP1_ForceReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ForceReset (uint32_t Periphs)

Function description

Force APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_MRBLE

Return values

- **None:**

Notes

- LL_APB2_GRP1_PERIPH_MRBLE

Reference Manual to LL API cross reference:

- APB2RSTR MRBLERST LL_APB2_GRP1_ForceReset

LL_APB2_GRP1_ReleaseReset

Function name

__STATIC_INLINE void LL_APB2_GRP1_ReleaseReset (uint32_t Periphs)

Function description

Release APB2 peripherals reset.

Parameters

- **Periphs:** This parameter can be a combination of the following values:
  – LL_APB2_GRP1_PERIPH_MRBLE

Return values

- **None:**

Notes

- LL_APB2_GRP1_PERIPH_MRBLE

Reference Manual to LL API cross reference:

- APB2RSTR MRBLERST LL_APB2_GRP1_ReleaseReset

45.2 BUS Firmware driver defines

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

45.2.1 BUS

BUS

AHB PERIPH

LL_AHB1_GRP1_PERIPH_ALL

LL_AHB1_GRP1_PERIPH_DMA

LL_AHB1_GRP1_PERIPH_GPIOA

LL_AHB1_GRP1_PERIPH_GPIOB

LL_AHB1_GRP1_PERIPH_CRC

LL_AHB1_GRP1_PERIPH_PKA

LL_AHB1_GRP1_PERIPH_RNG

APB0 PERIPH

LL_APB0_GRP1_PERIPH_TIM2

LL_APB0_GRP1_PERIPH_TIM16

LL_APB0_GRP1_PERIPH_TIM17

LL_APB0_GRP1_PERIPH_SYSCFG

LL_APB0_GRP1_PERIPH_RTC

LL_APB0_GRP1_PERIPH_WDG

APB1 PERIPH

LL_APB1_GRP1_PERIPH_ADCDIG

LL_APB1_GRP1_PERIPH_ADCANA

LL_APB1_GRP1_PERIPH_LPUART1

LL_APB1_GRP1_PERIPH_USART1

LL_APB1_GRP1_PERIPH_SPI3

LL_APB1_GRP1_PERIPH_I2C1

APB2 PERIPH

LL_APB2_GRP1_PERIPH_ALL

LL_APB2_GRP1_PERIPH_MRBLE

46 LL CORTEX Generic Driver

46.1 CORTEX Firmware driver API description

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

46.1.1 Detailed description of functions

LL_SYSTICK_Config

Function name

__STATIC_INLINE void LL_SYSTICK_Config (uint32_t ticks)

Function description

This function configures the Systick counter.

Parameters

- **ticks:** Number of ticks between two interrupts.

Return values

- **None.:**

Reference Manual to LL API cross reference:

- LL_SYSTICK_Config

LL_SYSTICK_Enable

Function name

__STATIC_INLINE void LL_SYSTICK_Enable (void)

Function description

This function enable the Systick counter.

Return values

- **None.:**

Reference Manual to LL API cross reference:

- STK_CTRL ENABLE LL_SYSTICK_Enable

LL_SYSTICK_Disable

Function name

__STATIC_INLINE void LL_SYSTICK_Disable (void)

Function description

This function disable the Systick counter.

Return values

- **None.:**

Reference Manual to LL API cross reference:

- STK_CTRL DISABLE LL_SYSTICK_Disable

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_SYSCLK

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_SYSCLK

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_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_GetConstant

Function name

__STATIC_INLINE uint32_t LL_CPUID_GetConstant (void)

Function description

Get Constant 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_GetConstant

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 or LL_MPU_REGION_SIZE_64KB or LL_MPU_REGION_SIZE_128KB or LL_MPU_REGION_SIZE_256KB or LL_MPU_REGION_SIZE_512KB or LL_MPU_REGION_SIZE_1MB or LL_MPU_REGION_SIZE_2MB or LL_MPU_REGION_SIZE_4MB or LL_MPU_REGION_SIZE_8MB or LL_MPU_REGION_SIZE_16MB or LL_MPU_REGION_SIZE_32MB or LL_MPU_REGION_SIZE_64MB or LL_MPU_REGION_SIZE_128MB or LL_MPU_REGION_SIZE_256MB or LL_MPU_REGION_SIZE_512MB or LL_MPU_REGION_SIZE_1GB or LL_MPU_REGION_SIZE_2GB or LL_MPU_REGION_SIZE_4GB
  - 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

46.2 CORTEX Firmware driver defines

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

46.2.1 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_SYSCLK

System 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

NVIC IRQ Priority

LL_NVIC_CRITICAL_PRIORITY

LL_NVIC_HIGH_PRIORITY

LL_NVIC_MED_PRIORITY

LL_NVIC_LOW_PRIORITY

47 LL CRC Generic Driver

47.1 CRC Firmware driver API description

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

47.1.1 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=0xFFFF FFFF

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

47.2 CRC Firmware driver defines

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

47.2.1 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

48 LL DMAMUX Generic Driver

48.1 DMAMUX Firmware driver API description

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

48.1.1 Detailed description of functions

LL_DMAMUX_SetRequestID

Function name

__STATIC_INLINE void LL_DMAMUX_SetRequestID (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
  - LL_DMAMUX_CHANNEL_7
- **Request:** This parameter can be one of the following values:
  - LL_DMAMUX_REQ_MEM2MEM
  - LL_DMAMUX_REQ_SPI3_RX
  - LL_DMAMUX_REQ_SPI3_TX
  - LL_DMAMUX_REQ_SPI1_RX
  - LL_DMAMUX_REQ_SPI1_TX
  - LL_DMAMUX_REQ_SPI2_RX
  - LL_DMAMUX_REQ_SPI2_TX
  - LL_DMAMUX_REQ_I2C1_RX
  - LL_DMAMUX_REQ_I2C1_TX
  - LL_DMAMUX_REQ_I2C2_RX
  - LL_DMAMUX_REQ_I2C2_TX
  - LL_DMAMUX_REQ_USART1_RX
  - LL_DMAMUX_REQ_USART1_TX
  - LL_DMAMUX_REQ_LPUART1_RX
  - LL_DMAMUX_REQ_LPUART1_TX
  - LL_DMAMUX_REQ_ADC1_DS
  - LL_DMAMUX_REQ_ADC1_DF

Return values

- **None:**

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 channel 1 to 8.

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 (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
  – LL_DMAMUX_CHANNEL_7

Return values

- **Returned:** value can be one of the following values:
  – LL_DMAMUX_REQ_MEM2MEM
  – LL_DMAMUX_REQ_SPI3_RX
  – LL_DMAMUX_REQ_SPI3_TX
  – LL_DMAMUX_REQ_SPI1_RX
  – LL_DMAMUX_REQ_SPI1_TX
  – LL_DMAMUX_REQ_SPI2_RX
  – LL_DMAMUX_REQ_SPI2_TX
  – LL_DMAMUX_REQ_I2C1_RX
  – LL_DMAMUX_REQ_I2C1_TX
  – LL_DMAMUX_REQ_I2C2_RX
  – LL_DMAMUX_REQ_I2C2_TX
  – LL_DMAMUX_REQ_USART1_RX
  – LL_DMAMUX_REQ_USART1_TX
  – LL_DMAMUX_REQ_LPUART1_RX
  – LL_DMAMUX_REQ_LPUART1_TX
  – LL_DMAMUX_REQ_ADC1_DS
  – LL_DMAMUX_REQ_ADC1_DF

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 channel 1 to 8.

Reference Manual to LL API cross reference:

- CxCR DMAREQ_ID LL_DMAMUX_GetRequestID

48.2 DMAMUX Firmware driver defines

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

48.2.1 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

LL_DMAMUX_CHANNEL_7

DMAMUX Channel 7 connected to DMA1 Channel 8

Transfer request

LL_DMAMUX_REQ_MEM2MEM

Memory to Memory transfer

LL_DMAMUX_REQ_SPI3_RX

DMAMUX SPI3 RX request

LL_DMAMUX_REQ_SPI3_TX

DMAMUX SPI3 TX request

LL_DMAMUX_REQ_I2C1_RX

DMAMUX I2C1 RX request

LL_DMAMUX_REQ_I2C1_TX

DMAMUX I2C1 TX request

LL_DMAMUX_REQ_USART1_RX

DMAMUX USART1 RX request

LL_DMAMUX_REQ_USART1_TX

DMAMUX USART1 TX request

LL_DMAMUX_REQ_LPUART1_RX

DMAMUX LPUART1 RX request

LL_DMAMUX_REQ_LPUART1_TX

DMAMUX LPUART1 TX request

LL_DMAMUX_REQ_ADC1_DS

DMAMUX ADC DS output request

LL_DMAMUX_REQ_TIM2_CH1

DMAMUX TIM2 CH1 request

LL_DMAMUX_REQ_TIM2_CH2

DMAMUX TIM2 CH3 request

LL_DMAMUX_REQ_TIM2_CH3

DMAMUX TIM2 CH3 request

LL_DMAMUX_REQ_TIM2_CH4

DMAMUX TIM2 CH4 request

LL_DMAMUX_REQ_TIM2_UP

DMAMUX TIM2 UP request

LL_DMAMUX_REQ_TIM16_CH1

DMAMUX TIM16 CH1 request

LL_DMAMUX_REQ_TIM16_UP

DMAMUX TIM16 UP request

LL_DMAMUX_REQ_TIM17_CH1

DMAMUX TIM17 CH1 request

LL_DMAMUX_REQ_TIM17_UP

DMAMUX TIM17 UP request

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

49 LL DMA Generic Driver

49.1 DMA Firmware driver registers structures

49.1.1 LL_DMA_InitTypeDef

LL_DMA_InitTypeDef is defined in the stm32wb0x_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 direction is 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()**.

49.2 DMA Firmware driver API description

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

49.2.1 Detailed description of functions

LL_DMA_EnableChannel

Function name

__STATIC_INLINE void LL_DMA_EnableChannel (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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_EnableChannel

LL_DMA_DisableChannel

Function name

__STATIC_INLINE void LL_DMA_DisableChannel (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
  - LL_DMA_CHANNEL_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_DisableChannel

LL_DMA_IsEnabledChannel

Function name

__STATIC_INLINE uint32_t LL_DMA_IsEnabledChannel (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
  - LL_DMA_CHANNEL_8

Return values

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

Reference Manual to LL API cross reference:

- CCR EN LL_DMA_IsEnabledChannel

LL_DMA_ConfigTransfer

Function name

__STATIC_INLINE void LL_DMA_ConfigTransfer (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
  – LL_DMA_CHANNEL_8
- **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:**

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 (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
  – LL_DMA_CHANNEL_8
- **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:**

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 (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
  – LL_DMA_CHANNEL_8

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

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 (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
  - LL_DMA_CHANNEL_8
- **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.

Reference Manual to LL API cross reference:

- CCR CIRC LL_DMA_SetMode

LL_DMA_GetMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMode (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
  - LL_DMA_CHANNEL_8

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_MODE_NORMAL
  – LL_DMA_MODE_CIRCULAR

Reference Manual to LL API cross reference:

- CCR CIRC LL_DMA_GetMode

LL_DMA_SetPeriphIncMode

Function name

__STATIC_INLINE void LL_DMA_SetPeriphIncMode (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
  – LL_DMA_CHANNEL_8
- **PeriphOrM2MSrcIncMode:** This parameter can be one of the following values:
  – LL_DMA_PERIPH_INCREMENT
  – LL_DMA_PERIPH_NOINCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR PINC LL_DMA_SetPeriphIncMode

LL_DMA_GetPeriphIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphIncMode (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
  – LL_DMA_CHANNEL_8

Return values

- **Returned:** value can be one of the following values:
  – LL_DMA_PERIPH_INCREMENT
  – LL_DMA_PERIPH_NOINCREMENT

Reference Manual to LL API cross reference:

- CCR PINC LL_DMA_GetPeriphIncMode

LL_DMA_SetMemoryIncMode

Function name

__STATIC_INLINE void LL_DMA_SetMemoryIncMode (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
  – LL_DMA_CHANNEL_8
- **MemoryOrM2MDstIncMode:** This parameter can be one of the following values:
  – LL_DMA_MEMORY_INCREMENT
  – LL_DMA_MEMORY_NOINCREMENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- CCR MINC LL_DMA_SetMemoryIncMode

LL_DMA_GetMemoryIncMode

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryIncMode (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
  - LL_DMA_CHANNEL_8

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MEMORY_INCREMENT
  - LL_DMA_MEMORY_NOINCREMENT

Reference Manual to LL API cross reference:

- CCR MINC LL_DMA_GetMemoryIncMode

LL_DMA_SetPeriphSize

Function name

__STATIC_INLINE void LL_DMA_SetPeriphSize (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
  - LL_DMA_CHANNEL_8
- **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:**

Reference Manual to LL API cross reference:

- CCR PSIZE LL_DMA_SetPeriphSize

LL_DMA_GetPeriphSize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphSize (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
  - LL_DMA_CHANNEL_8

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_PDATAALIGN_BYTE
  - LL_DMA_PDATAALIGN_HALFWORD
  - LL_DMA_PDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CCR PSIZE LL_DMA_GetPeriphSize

LL_DMA_SetMemorySize

Function name

__STATIC_INLINE void LL_DMA_SetMemorySize (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
  - LL_DMA_CHANNEL_8
- **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:**

Reference Manual to LL API cross reference:

- CCR MSIZE LL_DMA_SetMemorySize

LL_DMA_GetMemorySize

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemorySize (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
  - LL_DMA_CHANNEL_8

Return values

- **Returned:** value can be one of the following values:
  - LL_DMA_MDATAALIGN_BYTE
  - LL_DMA_MDATAALIGN_HALFWORD
  - LL_DMA_MDATAALIGN_WORD

Reference Manual to LL API cross reference:

- CCR MSIZE LL_DMA_GetMemorySize

LL_DMA_SetChannelPriorityLevel

Function name

__STATIC_INLINE void LL_DMA_SetChannelPriorityLevel (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
  - LL_DMA_CHANNEL_8
- **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:**

Reference Manual to LL API cross reference:

- CCR PL LL_DMA_SetChannelPriorityLevel

LL_DMA_GetChannelPriorityLevel

Function name

__STATIC_INLINE uint32_t LL_DMA_GetChannelPriorityLevel (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
  - LL_DMA_CHANNEL_8

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

Reference Manual to LL API cross reference:

- CCR PL LL_DMA_GetChannelPriorityLevel

LL_DMA_SetDataLength

Function name

__STATIC_INLINE void LL_DMA_SetDataLength (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
  - LL_DMA_CHANNEL_8
- **NbData:** Between Min_Data = 0 and Max_Data = 0x0000FFFF

Return values

- **None:**

Notes

- This action has no effect if channel is enabled.

Reference Manual to LL API cross reference:

- CNDTR NDT LL_DMA_SetDataLength

LL_DMA_GetDataLength

Function name

__STATIC_INLINE uint32_t LL_DMA_GetDataLength (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
  - LL_DMA_CHANNEL_8

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.

Reference Manual to LL API cross reference:

- CNDTR NDT LL_DMA_GetDataLength

LL_DMA_ConfigAddresses

Function name

__STATIC_INLINE void LL_DMA_ConfigAddresses (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
  - LL_DMA_CHANNEL_8
- **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 IP using DMA provides an API to get directly the register address (LL_PPP_DMA_GetRegAddr).

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 (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
  – LL_DMA_CHANNEL_8
- **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.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_SetMemoryAddress

LL_DMA_SetPeriphAddress

Function name

__STATIC_INLINE void LL_DMA_SetPeriphAddress (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
  – LL_DMA_CHANNEL_8
- **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.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_SetPeriphAddress

LL_DMA_GetMemoryAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetMemoryAddress (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
  - LL_DMA_CHANNEL_8

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.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_GetMemoryAddress

LL_DMA_GetPeriphAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetPeriphAddress (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
  - LL_DMA_CHANNEL_8

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.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_GetPeriphAddress

LL_DMA_SetM2MSrcAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MSrcAddress (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
  – LL_DMA_CHANNEL_8
- **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.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_SetM2MSrcAddress

LL_DMA_SetM2MDstAddress

Function name

__STATIC_INLINE void LL_DMA_SetM2MDstAddress (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
  - LL_DMA_CHANNEL_8
- **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.

Reference Manual to LL API cross reference:

- CMAR MA LL_DMA_SetM2MDstAddress

LL_DMA_GetM2MSrcAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MSrcAddress (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
  - LL_DMA_CHANNEL_8

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

Reference Manual to LL API cross reference:

- CPAR PA LL_DMA_GetM2MSrcAddress

LL_DMA_GetM2MDstAddress

Function name

__STATIC_INLINE uint32_t LL_DMA_GetM2MDstAddress (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
  – LL_DMA_CHANNEL_8

Return values

- **Between:** Min_Data = 0 and Max_Data = 0xFFFFFFFF

Notes

- Interface used for direction LL_DMA_DIRECTION_MEMORY_TO_MEMORY only.

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
  – LL_DMA_CHANNEL_8
- **Request:** This parameter can be a value of Transfer request

Return values

- **None:**

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 channel 1 to 8.

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
  - LL_DMA_CHANNEL_8

Return values

- **Returned:** This parameter can be a value of Transfer request

Notes

- DMAMUX channel 0 to 7 are mapped to DMA1 channel 1 to 8.

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 (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 (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 (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 (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 (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 (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 (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_GI8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_GI8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR GIF8 LL_DMA_IsActiveFlag_GI8

LL_DMA_IsActiveFlag_TC1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC1 (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 (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 (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 (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 (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 (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 (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_TC8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TC8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TCIF8 LL_DMA_IsActiveFlag_TC8

LL_DMA_IsActiveFlag_HT1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT1 (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 (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 (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 (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 (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 (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 (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_HT8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_HT8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR HTIF8 LL_DMA_IsActiveFlag_HT8

LL_DMA_IsActiveFlag_TE1

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE1 (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 (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 (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 (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 (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 (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 (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_IsActiveFlag_TE8

Function name

__STATIC_INLINE uint32_t LL_DMA_IsActiveFlag_TE8 (DMA_TypeDef * DMAx)

Function description

Get Channel 8 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TEIF8 LL_DMA_IsActiveFlag_TE8

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:**

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:**

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:**

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:**

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:**

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:**

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:**

Reference Manual to LL API cross reference:

- IFCR CGIF7 LL_DMA_ClearFlag_GI7

LL_DMA_ClearFlag_GI8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_GI8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 global interrupt flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CGIF8 LL_DMA_ClearFlag_GI8

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_TC8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TC8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 transfer complete flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTCIF8 LL_DMA_ClearFlag_TC8

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_HT8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_HT8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 half transfer flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CHTIF8 LL_DMA_ClearFlag_HT8

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_ClearFlag_TE8

Function name

__STATIC_INLINE void LL_DMA_ClearFlag_TE8 (DMA_TypeDef * DMAx)

Function description

Clear Channel 8 transfer error flag.

Parameters

- **DMAx:** DMAx Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- IFCR CTEIF8 LL_DMA_ClearFlag_TE8

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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

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 (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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

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 (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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

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 (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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

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 (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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

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 (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
  – LL_DMA_CHANNEL_8

Return values

- **None:**

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 (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
  - LL_DMA_CHANNEL_8

Return values

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

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 (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
  - LL_DMA_CHANNEL_8

Return values

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

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 (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
  - LL_DMA_CHANNEL_8

Return values

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

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
  - LL_DMA_CHANNEL_8
- **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

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_8
  – LL_DMA_CHANNEL_ALL

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: DMA registers are de-initialized
  – ERROR: DMA registers are not de-initialized

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:**

49.3 DMA Firmware driver defines

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

49.3.1 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_8

DMA Channel 8

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

LL_DMA_IFCR_CGIF8

Channel 8 global flag

LL_DMA_IFCR_CTCIF8

Channel 8 transfer complete flag

LL_DMA_IFCR_CHTIF8

Channel 8 half transfer flag

LL_DMA_IFCR_CTEIF8

Channel 8 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

LL_DMA_ISR_GIF8

Channel 8 global flag

LL_DMA_ISR_TCIF8

Channel 8 transfer complete flag

LL_DMA_ISR_HTIF8

Channel 8 half transfer flag

LL_DMA_ISR_TEIF8

Channel 8 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

50 LL GPIO Generic Driver

50.1 GPIO Firmware driver registers structures

50.1.1 LL_GPIO_InitTypeDef

LL_GPIO_InitTypeDef is defined in the stm32wb0x_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()**.

50.2 GPIO Firmware driver API description

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

50.2.1 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 (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 (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 (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 (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

Return values

- **None:**

Notes

- Possible values are from AF0 to AF7 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 (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

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

Return values

- **None:**

Notes

- Possible values are from AF0 to AF7 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 (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

Notes

- Possible values are from AF0 to AF7 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 (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 (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 (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 (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 (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 (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 (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

Notes

- The application must ensure that the PWRC_CR1_APC bit is reset via LL PWRC services.

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:**

50.3 GPIO Firmware driver defines

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

50.3.1 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

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

51 LL I2C Generic Driver

51.1 I2C Firmware driver registers structures

51.1.1 LL_I2C_InitTypeDef

LL_I2C_InitTypeDef is defined in the stm32wb0x_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()**.

51.2 I2C Firmware driver API description

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

51.2.1 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_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:**

51.3 I2C Firmware driver defines

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

51.3.1 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

52 LL I2S Generic Driver

52.1 I2S Firmware driver registers structures

52.1.1 LL_I2S_InitTypeDef

LL_I2S_InitTypeDef is defined in the stm32wb0x_ll_i2s.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()**.

52.2 I2S Firmware driver API description

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

52.2.1 Detailed description of functions

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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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).

52.3 I2S Firmware driver defines

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

52.3.1 I2S

I2S

Audio Frequency

LL_I2S_AUDIOFREQ_192K

Audio Frequency configuration 192000 Hz

LL_I2S_AUDIOFREQ_96K

Audio Frequency configuration 96000 Hz

LL_I2S_AUDIOFREQ_48K

Audio Frequency configuration 48000 Hz

LL_I2S_AUDIOFREQ_44K

Audio Frequency configuration 44100 Hz

LL_I2S_AUDIOFREQ_32K

Audio Frequency configuration 32000 Hz

LL_I2S_AUDIOFREQ_22K

Audio Frequency configuration 22050 Hz

LL_I2S_AUDIOFREQ_16K

Audio Frequency configuration 16000 Hz

LL_I2S_AUDIOFREQ_11K

Audio Frequency configuration 11025 Hz

LL_I2S_AUDIOFREQ_8K

Audio Frequency configuration 8000 Hz

LL_I2S_AUDIOFREQ_DEFAULT

Audio Freq not specified. Register I2SDIV = 2

LL_I2S_AUDIOFREQ_192K

Audio Frequency configuration 192000 Hz

LL_I2S_AUDIOFREQ_96K

Audio Frequency configuration 96000 Hz

LL_I2S_AUDIOFREQ_48K

Audio Frequency configuration 48000 Hz

LL_I2S_AUDIOFREQ_44K

Audio Frequency configuration 44100 Hz

LL_I2S_AUDIOFREQ_32K

Audio Frequency configuration 32000 Hz

LL_I2S_AUDIOFREQ_22K

Audio Frequency configuration 22050 Hz

LL_I2S_AUDIOFREQ_16K

Audio Frequency configuration 16000 Hz

LL_I2S_AUDIOFREQ_11K

Audio Frequency configuration 11025 Hz

LL_I2S_AUDIOFREQ_8K

Audio Frequency configuration 8000 Hz

LL_I2S_AUDIOFREQ_DEFAULT

Audio Freq not specified. Register I2SDIV = 2

Data format

LL_I2S_DATAFORMAT_16B

Data length 16 bits, Channel length 16bit

LL_I2S_DATAFORMAT_16B_EXTENDED

Data length 16 bits, Channel length 32bit

LL_I2S_DATAFORMAT_24B

Data length 24 bits, Channel length 32bit

LL_I2S_DATAFORMAT_32B

Data length 16 bits, Channel length 32bit

LL_I2S_DATAFORMAT_16B

Data length 16 bits, Channel length 16bit

LL_I2S_DATAFORMAT_16B_EXTENDED

Data length 16 bits, Channel length 32bit

LL_I2S_DATAFORMAT_24B

Data length 24 bits, Channel length 32bit

LL_I2S_DATAFORMAT_32B

Data length 16 bits, Channel length 32bit

Get Flags Defines

LL_I2S_SR_RXNE

Rx buffer not empty flag

LL_I2S_SR_TXE

Tx buffer empty flag

LL_I2S_SR_BSY

Busy flag

LL_I2S_SR_UDR

Underrun flag

LL_I2S_SR_OVR

Overrun flag

LL_I2S_SR_FRE

TI mode frame format error flag

LL_I2S_SR_RXNE

Rx buffer not empty flag

LL_I2S_SR_TXE

Tx buffer empty flag

LL_I2S_SR_BSY

Busy flag

LL_I2S_SR_UDR

Underrun flag

LL_I2S_SR_OVR

Overrun flag

LL_I2S_SR_FRE

TI mode frame format error flag

MCLK Output

LL_I2S_MCLK_OUTPUT_DISABLE

Master clock output is disabled

LL_I2S_MCLK_OUTPUT_ENABLE

Master clock output is enabled

LL_I2S_MCLK_OUTPUT_DISABLE

Master clock output is disabled

LL_I2S_MCLK_OUTPUT_ENABLE

Master clock output is enabled

Operation Mode

LL_I2S_MODE_SLAVE_TX

Slave Tx configuration

LL_I2S_MODE_SLAVE_RX

Slave Rx configuration

LL_I2S_MODE_MASTER_TX

Master Tx configuration

LL_I2S_MODE_MASTER_RX

Master Rx configuration

LL_I2S_MODE_SLAVE_TX

Slave Tx configuration

LL_I2S_MODE_SLAVE_RX

Slave Rx configuration

LL_I2S_MODE_MASTER_TX

Master Tx configuration

LL_I2S_MODE_MASTER_RX

Master Rx configuration

Clock Polarity

LL_I2S_POLARITY_LOW

Clock steady state is low level

LL_I2S_POLARITY_HIGH

Clock steady state is high level

LL_I2S_POLARITY_LOW

Clock steady state is low level

LL_I2S_POLARITY_HIGH

Clock steady state is high level

Prescaler Factor

LL_I2S_PRESCALER_PARITY_EVEN

Odd factor: Real divider value is = I2SDIV * 2

LL_I2S_PRESCALER_PARITY_ODD

Odd factor: Real divider value is = (I2SDIV * 2)+1

LL_I2S_PRESCALER_PARITY_EVEN

Odd factor: Real divider value is = I2SDIV * 2

LL_I2S_PRESCALER_PARITY_ODD

Odd factor: Real divider value is = (I2SDIV * 2)+1

I2s Standard

LL_I2S_STANDARD_PHILIPS

I2S standard philips

LL_I2S_STANDARD_MSB

MSB justified standard (left justified)

LL_I2S_STANDARD_LSB

LSB justified standard (right justified)

LL_I2S_STANDARD_PCM_SHORT

PCM standard, short frame synchronization

LL_I2S_STANDARD_PCM_LONG

PCM standard, long frame synchronization

LL_I2S_STANDARD_PHILIPS

I2S standard philips

LL_I2S_STANDARD_MSB

MSB justified standard (left justified)

LL_I2S_STANDARD_LSB

LSB justified standard (right justified)

LL_I2S_STANDARD_PCM_SHORT

PCM standard, short frame synchronization

LL_I2S_STANDARD_PCM_LONG

PCM standard, long frame synchronization

Common Write and read registers Macros

LL_I2S_WriteReg

Description:

- Write a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2S_ReadReg

Description:

- Read a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be read

Return value:

- Register: value

LL_I2S_WriteReg

Description:

- Write a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_I2S_ReadReg

Description:

- Read a value in I2S register.

Parameters:

- __INSTANCE__: I2S Instance
- __REG__: Register to be read

Return value:

- Register: value

53 LL IWDG Generic Driver

53.1 IWDG Firmware driver API description

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

53.1.1 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 (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 (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 (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 (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 (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 (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 (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

53.2 IWDG Firmware driver defines

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

53.2.1 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

54 LL LPUART Generic Driver

54.1 LPUART Firmware driver registers structures

54.1.1 LL_LPUART_InitTypeDef

LL_LPUART_InitTypeDef is defined in the stm32wb0x_ll_lpuart.h

Data Fields

- *uint32_t PrescalerValue*
- *uint32_t BaudRate*
- *uint32_t DataWidth*
- *uint32_t StopBits*
- *uint32_t Parity*
- *uint32_t TransferDirection*
- *uint32_t HardwareFlowControl*

Field Documentation

- ***uint32_t LL_LPUART_InitTypeDef::PrescalerValue***
  Specifies the Prescaler to compute the communication baud rate. This parameter can be a value of *LPUART_LL_EC_PRESCALER*.This feature can be modified afterwards using unitary function **LL_LPUART_SetPrescaler()**.
- ***uint32_t LL_LPUART_InitTypeDef::BaudRate***
  This field defines expected LPUART communication baud rate.This feature can be modified afterwards using unitary function **LL_LPUART_SetBaudRate()**.
- ***uint32_t LL_LPUART_InitTypeDef::DataWidth***
  Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of *LPUART_LL_EC_DATAWIDTH*.This feature can be modified afterwards using unitary function **LL_LPUART_SetDataWidth()**.
- ***uint32_t LL_LPUART_InitTypeDef::StopBits***
  Specifies the number of stop bits transmitted. This parameter can be a value of *LPUART_LL_EC_STOPBITS*.This feature can be modified afterwards using unitary function **LL_LPUART_SetStopBitsLength()**.
- ***uint32_t LL_LPUART_InitTypeDef::Parity***
  Specifies the parity mode. This parameter can be a value of *LPUART_LL_EC_PARITY*.This feature can be modified afterwards using unitary function **LL_LPUART_SetParity()**.
- ***uint32_t LL_LPUART_InitTypeDef::TransferDirection***
  Specifies whether the Receive and/or Transmit mode is enabled or disabled. This parameter can be a value of *LPUART_LL_EC_DIRECTION*.This feature can be modified afterwards using unitary function **LL_LPUART_SetTransferDirection()**.
- ***uint32_t LL_LPUART_InitTypeDef::HardwareFlowControl***
  Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of *LPUART_LL_EC_HWCONTROL*.This feature can be modified afterwards using unitary function **LL_LPUART_SetHWFlowCtrl()**.

54.2 LPUART Firmware driver API description

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

54.2.1 Detailed description of functions

LL_LPUART_Enable

Function name

__STATIC_INLINE void LL_LPUART_Enable (USART_TypeDef * LPUARTx)

Function description

LPUART Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Enable

LL_LPUART_Disable

Function name

__STATIC_INLINE void LL_LPUART_Disable (USART_TypeDef * LPUARTx)

Function description

LPUART Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When LPUART is disabled, LPUART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the LPUART is kept, but all the status flags, in the LPUARTx_ISR are set to their default values.
- In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the LPUART_ISR to be set before resetting the UE bit. The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_Disable

LL_LPUART_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabled (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 UE LL_LPUART_IsEnabled

LL_LPUART_EnableFIFO

Function name

__STATIC_INLINE void LL_LPUART_EnableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_EnableFIFO

LL_LPUART_DisableFIFO

Function name

__STATIC_INLINE void LL_LPUART_DisableFIFO (USART_TypeDef * LPUARTx)

Function description

FIFO Mode Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_DisableFIFO

LL_LPUART_IsEnabledFIFO

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledFIFO (const USART_TypeDef * LPUARTx)

Function description

Indicate if FIFO Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 FIFOEN LL_LPUART_IsEnabledFIFO

LL_LPUART_SetTXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetTXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure TX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  – LL_LPUART_FIFOTHRESHOLD_1_8
  – LL_LPUART_FIFOTHRESHOLD_1_4
  – LL_LPUART_FIFOTHRESHOLD_1_2
  – LL_LPUART_FIFOTHRESHOLD_3_4
  – LL_LPUART_FIFOTHRESHOLD_7_8
  – LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_SetTXFIFOThreshold

LL_LPUART_GetTXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return TX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_FIFOTHRESHOLD_1_8
  – LL_LPUART_FIFOTHRESHOLD_1_4
  – LL_LPUART_FIFOTHRESHOLD_1_2
  – LL_LPUART_FIFOTHRESHOLD_3_4
  – LL_LPUART_FIFOTHRESHOLD_7_8
  – LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_GetTXFIFOThreshold

LL_LPUART_SetRXFIFOThreshold

Function name

__STATIC_INLINE void LL_LPUART_SetRXFIFOThreshold (USART_TypeDef * LPUARTx, uint32_t Threshold)

Function description

Configure RX FIFO Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **Threshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_SetRXFIFOThreshold

LL_LPUART_GetRXFIFOThreshold

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXFIFOThreshold (const USART_TypeDef * LPUARTx)

Function description

Return RX FIFO Threshold Configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Reference Manual to LL API cross reference:

- CR3 RXFTCFG LL_LPUART_GetRXFIFOThreshold

LL_LPUART_ConfigFIFOsThreshold

Function name

__STATIC_INLINE void LL_LPUART_ConfigFIFOsThreshold (USART_TypeDef * LPUARTx, uint32_t TXThreshold, uint32_t RXThreshold)

Function description

Configure TX and RX FIFOs Threshold.

Parameters

- **LPUARTx:** LPUART Instance
- **TXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8
- **RXThreshold:** This parameter can be one of the following values:
  - LL_LPUART_FIFOTHRESHOLD_1_8
  - LL_LPUART_FIFOTHRESHOLD_1_4
  - LL_LPUART_FIFOTHRESHOLD_1_2
  - LL_LPUART_FIFOTHRESHOLD_3_4
  - LL_LPUART_FIFOTHRESHOLD_7_8
  - LL_LPUART_FIFOTHRESHOLD_8_8

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTCFG LL_LPUART_ConfigFIFOsThreshold
- CR3 RXFTCFG LL_LPUART_ConfigFIFOsThreshold

LL_LPUART_EnableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_EnableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART enabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is enabled, LPUART is able to wake up the MCU from Stop mode, provided that LPUART clock selection is HSI or LSE in RCC.

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_EnableInStopMode

LL_LPUART_DisableInStopMode

Function name

__STATIC_INLINE void LL_LPUART_DisableInStopMode (USART_TypeDef * LPUARTx)

Function description

LPUART disabled in STOP Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When this function is disabled, LPUART is not able to wake up the MCU from Stop mode

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_DisableInStopMode

LL_LPUART_IsEnabledInStopMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledInStopMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if LPUART is enabled in STOP Mode (able to wake up MCU from Stop mode or not)

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 UESM LL_LPUART_IsEnabledInStopMode

LL_LPUART_EnableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Enable (Receiver is enabled and begins searching for a start bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_EnableDirectionRx

LL_LPUART_DisableDirectionRx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionRx (USART_TypeDef * LPUARTx)

Function description

Receiver Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_DisableDirectionRx

LL_LPUART_EnableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_EnableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Enable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_EnableDirectionTx

LL_LPUART_DisableDirectionTx

Function name

__STATIC_INLINE void LL_LPUART_DisableDirectionTx (USART_TypeDef * LPUARTx)

Function description

Transmitter Disable.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TE LL_LPUART_DisableDirectionTx

LL_LPUART_SetTransferDirection

Function name

__STATIC_INLINE void LL_LPUART_SetTransferDirection (USART_TypeDef * LPUARTx, uint32_t TransferDirection)

Function description

Configure simultaneously enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance
- **TransferDirection:** This parameter can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_SetTransferDirection
- CR1 TE LL_LPUART_SetTransferDirection

LL_LPUART_GetTransferDirection

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferDirection (const USART_TypeDef * LPUARTx)

Function description

Return enabled/disabled states of Transmitter and Receiver.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DIRECTION_NONE
  - LL_LPUART_DIRECTION_RX
  - LL_LPUART_DIRECTION_TX
  - LL_LPUART_DIRECTION_TX_RX

Reference Manual to LL API cross reference:

- CR1 RE LL_LPUART_GetTransferDirection
- CR1 TE LL_LPUART_GetTransferDirection

LL_LPUART_SetParity

Function name

__STATIC_INLINE void LL_LPUART_SetParity (USART_TypeDef * LPUARTx, uint32_t Parity)

Function description

Configure Parity (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance
- **Parity:** This parameter can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Return values

- **None:**

Notes

- This function selects if hardware parity control (generation and detection) is enabled or disabled. When the parity control is enabled (Odd or Even), computed parity bit is inserted at the MSB position (depending on data width) and parity is checked on the received data.

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_SetParity
- CR1 PCE LL_LPUART_SetParity

LL_LPUART_GetParity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetParity (const USART_TypeDef * LPUARTx)

Function description

Return Parity configuration (enabled/disabled and parity mode if enabled)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PARITY_NONE
  - LL_LPUART_PARITY_EVEN
  - LL_LPUART_PARITY_ODD

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_GetParity
- CR1 PCE LL_LPUART_GetParity

LL_LPUART_SetWakeUpMethod

Function name

__STATIC_INLINE void LL_LPUART_SetWakeUpMethod (USART_TypeDef * LPUARTx, uint32_t Method)

Function description

Set Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance
- **Method:** This parameter can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_SetWakeUpMethod

LL_LPUART_GetWakeUpMethod

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWakeUpMethod (const USART_TypeDef * LPUARTx)

Function description

Return Receiver Wake Up method from Mute mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_IDLELINE
  - LL_LPUART_WAKEUP_ADDRESSMARK

Reference Manual to LL API cross reference:

- CR1 WAKE LL_LPUART_GetWakeUpMethod

LL_LPUART_SetDataWidth

Function name

__STATIC_INLINE void LL_LPUART_SetDataWidth (USART_TypeDef * LPUARTx, uint32_t DataWidth)

Function description

Set Word length (nb of data bits, excluding start and stop bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_SetDataWidth

LL_LPUART_GetDataWidth

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDataWidth (const USART_TypeDef * LPUARTx)

Function description

Return Word length (i.e.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_DATAWIDTH_7B
  - LL_LPUART_DATAWIDTH_8B
  - LL_LPUART_DATAWIDTH_9B

Reference Manual to LL API cross reference:

- CR1 M LL_LPUART_GetDataWidth

LL_LPUART_EnableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_EnableMuteMode (USART_TypeDef * LPUARTx)

Function description

Allow switch between Mute Mode and Active mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_EnableMuteMode

LL_LPUART_DisableMuteMode

Function name

__STATIC_INLINE void LL_LPUART_DisableMuteMode (USART_TypeDef * LPUARTx)

Function description

Prevent Mute Mode use.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_DisableMuteMode

LL_LPUART_IsEnabledMuteMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledMuteMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if switch between Mute Mode and Active mode is allowed.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 MME LL_LPUART_IsEnabledMuteMode

LL_LPUART_SetPrescaler

Function name

__STATIC_INLINE void LL_LPUART_SetPrescaler (USART_TypeDef * LPUARTx, uint32_t PrescalerValue)

Function description

Configure Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Return values

- **None:**

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_SetPrescaler

LL_LPUART_GetPrescaler

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetPrescaler (const USART_TypeDef * LPUARTx)

Function description

Retrieve the Clock source prescaler for baudrate generator and oversampling.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256

Reference Manual to LL API cross reference:

- PRESC PRESCALER LL_LPUART_GetPrescaler

LL_LPUART_SetStopBitsLength

Function name

__STATIC_INLINE void LL_LPUART_SetStopBitsLength (USART_TypeDef * LPUARTx, uint32_t StopBits)

Function description

Set the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance
- **StopBits:** This parameter can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_SetStopBitsLength

LL_LPUART_GetStopBitsLength

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetStopBitsLength (const USART_TypeDef * LPUARTx)

Function description

Retrieve the length of the stop bits.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_STOPBITS_1
  - LL_LPUART_STOPBITS_2

Reference Manual to LL API cross reference:

- CR2 STOP LL_LPUART_GetStopBitsLength

LL_LPUART_ConfigCharacter

Function name

__STATIC_INLINE void LL_LPUART_ConfigCharacter (USART_TypeDef * LPUARTx, uint32_t DataWidth, uint32_t Parity, uint32_t StopBits)

Function description

Configure Character frame format (Datawidth, Parity control, Stop Bits)

Parameters

- **LPUARTx:** LPUART Instance
- **DataWidth:** This parameter can be one of the following values:
  – LL_LPUART_DATAWIDTH_7B
  – LL_LPUART_DATAWIDTH_8B
  – LL_LPUART_DATAWIDTH_9B
- **Parity:** This parameter can be one of the following values:
  – LL_LPUART_PARITY_NONE
  – LL_LPUART_PARITY_EVEN
  – LL_LPUART_PARITY_ODD
- **StopBits:** This parameter can be one of the following values:
  – LL_LPUART_STOPBITS_1
  – LL_LPUART_STOPBITS_2

Return values

- **None:**

Notes

- Call of this function is equivalent to following function call sequence : Data Width configuration using LL_LPUART_SetDataWidth() functionParity Control and mode configuration using LL_LPUART_SetParity() functionStop bits configuration using LL_LPUART_SetStopBitsLength() function

Reference Manual to LL API cross reference:

- CR1 PS LL_LPUART_ConfigCharacter
- CR1 PCE LL_LPUART_ConfigCharacter
- CR1 M LL_LPUART_ConfigCharacter
- CR2 STOP LL_LPUART_ConfigCharacter

LL_LPUART_SetTXRXSwap

Function name

__STATIC_INLINE void LL_LPUART_SetTXRXSwap (USART_TypeDef * LPUARTx, uint32_t SwapConfig)

Function description

Configure TX/RX pins swapping setting.

Parameters

- **LPUARTx:** LPUART Instance
- **SwapConfig:** This parameter can be one of the following values:
  – LL_LPUART_TXRX_STANDARD
  – LL_LPUART_TXRX_SWAPPED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_SetTXRXSwap

LL_LPUART_GetTXRXSwap

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXRXSwap (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX/RX pins swapping configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_TXRX_STANDARD
  – LL_LPUART_TXRX_SWAPPED

Reference Manual to LL API cross reference:

- CR2 SWAP LL_LPUART_GetTXRXSwap

LL_LPUART_SetRXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetRXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure RX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  – LL_LPUART_RXPIN_LEVEL_STANDARD
  – LL_LPUART_RXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_SetRXPinLevel

LL_LPUART_GetRXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetRXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve RX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_RXPIN_LEVEL_STANDARD
  – LL_LPUART_RXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 RXINV LL_LPUART_GetRXPinLevel

LL_LPUART_SetTXPinLevel

Function name

__STATIC_INLINE void LL_LPUART_SetTXPinLevel (USART_TypeDef * LPUARTx, uint32_t PinInvMethod)

Function description

Configure TX pin active level logic.

Parameters

- **LPUARTx:** LPUART Instance
- **PinInvMethod:** This parameter can be one of the following values:
  – LL_LPUART_TXPIN_LEVEL_STANDARD
  – LL_LPUART_TXPIN_LEVEL_INVERTED

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_SetTXPinLevel

LL_LPUART_GetTXPinLevel

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTXPinLevel (const USART_TypeDef * LPUARTx)

Function description

Retrieve TX pin active level logic configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_TXPIN_LEVEL_STANDARD
  – LL_LPUART_TXPIN_LEVEL_INVERTED

Reference Manual to LL API cross reference:

- CR2 TXINV LL_LPUART_GetTXPinLevel

LL_LPUART_SetBinaryDataLogic

Function name

__STATIC_INLINE void LL_LPUART_SetBinaryDataLogic (USART_TypeDef * LPUARTx, uint32_t DataLogic)

Function description

Configure Binary data logic.

Parameters

- **LPUARTx:** LPUART Instance
- **DataLogic:** This parameter can be one of the following values:
  – LL_LPUART_BINARY_LOGIC_POSITIVE
  – LL_LPUART_BINARY_LOGIC_NEGATIVE

Return values

- **None:**

Notes

- Allow to define how Logical data from the data register are send/received : either in positive/direct logic (1=H, 0=L) or in negative/inverse logic (1=L, 0=H)

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_SetBinaryDataLogic

LL_LPUART_GetBinaryDataLogic

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBinaryDataLogic (const USART_TypeDef * LPUARTx)

Function description

Retrieve Binary data configuration.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BINARY_LOGIC_POSITIVE
  - LL_LPUART_BINARY_LOGIC_NEGATIVE

Reference Manual to LL API cross reference:

- CR2 DATAINV LL_LPUART_GetBinaryDataLogic

LL_LPUART_SetTransferBitOrder

Function name

__STATIC_INLINE void LL_LPUART_SetTransferBitOrder (USART_TypeDef * LPUARTx, uint32_t BitOrder)

Function description

Configure transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance
- **BitOrder:** This parameter can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Return values

- **None:**

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_SetTransferBitOrder

LL_LPUART_GetTransferBitOrder

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetTransferBitOrder (const USART_TypeDef * LPUARTx)

Function description

Return transfer bit order (either Less or Most Significant Bit First)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_BITORDER_LSBFIRST
  - LL_LPUART_BITORDER_MSBFIRST

Notes

- MSB First means data is transmitted/received with the MSB first, following the start bit. LSB First means data is transmitted/received with data bit 0 first, following the start bit.

Reference Manual to LL API cross reference:

- CR2 MSBFIRST LL_LPUART_GetTransferBitOrder

LL_LPUART_ConfigNodeAddress

Function name

__STATIC_INLINE void LL_LPUART_ConfigNodeAddress (USART_TypeDef * LPUARTx, uint32_t AddressLen, uint32_t NodeAddress)

Function description

Set Address of the LPUART node.

Parameters

- **LPUARTx:** LPUART Instance
- **AddressLen:** This parameter can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B
- **NodeAddress:** 4 or 7 bit Address of the LPUART node.

Return values

- **None:**

Notes

- This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with address mark detection.
- 4bits address node is used when 4-bit Address Detection is selected in ADDM7. (b7-b4 should be set to 0) 8bits address node is used when 7-bit Address Detection is selected in ADDM7. (This is used in multiprocessor communication during Mute mode or Stop mode, for wake up with 7-bit address mark detection. The MSB of the character sent by the transmitter should be equal to 1. It may also be used for character detection during normal reception, Mute mode inactive (for example, end of block detection in ModBus protocol). In this case, the whole received character (8-bit) is compared to the ADD[7:0] value and CMF flag is set on match)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_ConfigNodeAddress
- CR2 ADDM7 LL_LPUART_ConfigNodeAddress

LL_LPUART_GetNodeAddress

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddress (const USART_TypeDef * LPUARTx)

Function description

Return 8 bit Address of the LPUART node as set in ADD field of CR2.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Address:** of the LPUART node (Value between Min_Data=0 and Max_Data=255)

Notes

- If 4-bit Address Detection is selected in ADDM7, only 4bits (b3-b0) of returned value are relevant (b31-b4 are not relevant) If 7-bit Address Detection is selected in ADDM7, only 8bits (b7-b0) of returned value are relevant (b31-b8 are not relevant)

Reference Manual to LL API cross reference:

- CR2 ADD LL_LPUART_GetNodeAddress

LL_LPUART_GetNodeAddressLen

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetNodeAddressLen (const USART_TypeDef * LPUARTx)

Function description

Return Length of Node Address used in Address Detection mode (7-bit or 4-bit)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_ADDRESS_DETECT_4B
  - LL_LPUART_ADDRESS_DETECT_7B

Reference Manual to LL API cross reference:

- CR2 ADDM7 LL_LPUART_GetNodeAddressLen

LL_LPUART_EnableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_EnableRTSHWFlowCtrl

LL_LPUART_DisableRTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableRTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable RTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_DisableRTSHWFlowCtrl

LL_LPUART_EnableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_EnableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Enable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_EnableCTSHWFlowCtrl

LL_LPUART_DisableCTSHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_DisableCTSHWFlowCtrl (USART_TypeDef * LPUARTx)

Function description

Disable CTS HW Flow Control.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSE LL_LPUART_DisableCTSHWFlowCtrl

LL_LPUART_SetHWFlowCtrl

Function name

__STATIC_INLINE void LL_LPUART_SetHWFlowCtrl (USART_TypeDef * LPUARTx, uint32_t HardwareFlowControl)

Function description

Configure HW Flow Control mode (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance
- **HardwareFlowControl:** This parameter can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_SetHWFlowCtrl
- CR3 CTSE LL_LPUART_SetHWFlowCtrl

LL_LPUART_GetHWFlowCtrl

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetHWFlowCtrl (const USART_TypeDef * LPUARTx)

Function description

Return HW Flow Control configuration (both CTS and RTS)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_HWCONTROL_NONE
  - LL_LPUART_HWCONTROL_RTS
  - LL_LPUART_HWCONTROL_CTS
  - LL_LPUART_HWCONTROL_RTS_CTS

Reference Manual to LL API cross reference:

- CR3 RTSE LL_LPUART_GetHWFlowCtrl
- CR3 CTSE LL_LPUART_GetHWFlowCtrl

LL_LPUART_EnableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_EnableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Enable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_EnableOverrunDetect

LL_LPUART_DisableOverrunDetect

Function name

__STATIC_INLINE void LL_LPUART_DisableOverrunDetect (USART_TypeDef * LPUARTx)

Function description

Disable Overrun detection.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_DisableOverrunDetect

LL_LPUART_IsEnabledOverrunDetect

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledOverrunDetect (const USART_TypeDef * LPUARTx)

Function description

Indicate if Overrun detection is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 OVRDIS LL_LPUART_IsEnabledOverrunDetect

LL_LPUART_SetWKUPType

Function name

__STATIC_INLINE void LL_LPUART_SetWKUPType (USART_TypeDef * LPUARTx, uint32_t Type)

Function description

Select event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Type:** This parameter can be one of the following values:
  – LL_LPUART_WAKEUP_ON_ADDRESS
  – LL_LPUART_WAKEUP_ON_STARTBIT
  – LL_LPUART_WAKEUP_ON_RXNE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_SetWKUPType

LL_LPUART_GetWKUPType

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetWKUPType (const USART_TypeDef * LPUARTx)

Function description

Return event type for Wake UP Interrupt Flag (WUS[1:0] bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  - LL_LPUART_WAKEUP_ON_ADDRESS
  - LL_LPUART_WAKEUP_ON_STARTBIT
  - LL_LPUART_WAKEUP_ON_RXNE

Reference Manual to LL API cross reference:

- CR3 WUS LL_LPUART_GetWKUPType

LL_LPUART_SetBaudRate

Function name

__STATIC_INLINE void LL_LPUART_SetBaudRate (USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue, uint32_t BaudRate)

Function description

Configure LPUART BRR register for achieving expected Baud Rate value.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- **BaudRate:** Baud Rate

Return values

- **None:**

Notes

- Compute and set LPUARTDIV value in BRR Register (full BRR content) according to used Peripheral Clock and expected Baud Rate values
- Peripheral clock and Baud Rate values provided as function parameters should be valid (Baud rate value != 0).
- Provided that LPUARTx_BRR must be > = 0x300 and LPUART_BRR is 20-bit, a care should be taken when generating high baud rates using high PeriphClk values. PeriphClk must be in the range [3 x BaudRate, 4096 x BaudRate].

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_SetBaudRate

LL_LPUART_GetBaudRate

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetBaudRate (const USART_TypeDef * LPUARTx, uint32_t PeriphClk, uint32_t PrescalerValue)

Function description

Return current Baud Rate value, according to LPUARTDIV present in BRR register (full BRR content), and to used Peripheral Clock values.

Parameters

- **LPUARTx:** LPUART Instance
- **PeriphClk:** Peripheral Clock
- **PrescalerValue:** This parameter can be one of the following values:
  – LL_LPUART_PRESCALER_DIV1
  – LL_LPUART_PRESCALER_DIV2
  – LL_LPUART_PRESCALER_DIV4
  – LL_LPUART_PRESCALER_DIV6
  – LL_LPUART_PRESCALER_DIV8
  – LL_LPUART_PRESCALER_DIV10
  – LL_LPUART_PRESCALER_DIV12
  – LL_LPUART_PRESCALER_DIV16
  – LL_LPUART_PRESCALER_DIV32
  – LL_LPUART_PRESCALER_DIV64
  – LL_LPUART_PRESCALER_DIV128
  – LL_LPUART_PRESCALER_DIV256

Return values

- **Baud:** Rate

Notes

- In case of non-initialized or invalid value stored in BRR register, value 0 will be returned.

Reference Manual to LL API cross reference:

- BRR BRR LL_LPUART_GetBaudRate

LL_LPUART_EnableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_EnableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Enable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_EnableHalfDuplex

LL_LPUART_DisableHalfDuplex

Function name

__STATIC_INLINE void LL_LPUART_DisableHalfDuplex (USART_TypeDef * LPUARTx)

Function description

Disable Single Wire Half-Duplex mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_DisableHalfDuplex

LL_LPUART_IsEnabledHalfDuplex

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledHalfDuplex (const USART_TypeDef * LPUARTx)

Function description

Indicate if Single Wire Half-Duplex mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 HDSEL LL_LPUART_IsEnabledHalfDuplex

LL_LPUART_SetDEDeassertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEDeassertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEDT (Driver Enable De-Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_SetDEDeassertionTime

LL_LPUART_GetDEDeassertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEDeassertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEDT (Driver Enable De-Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : c

Reference Manual to LL API cross reference:

- CR1 DEDT LL_LPUART_GetDEDeassertionTime

LL_LPUART_SetDEAssertionTime

Function name

__STATIC_INLINE void LL_LPUART_SetDEAssertionTime (USART_TypeDef * LPUARTx, uint32_t Time)

Function description

Set DEAT (Driver Enable Assertion Time), Time value expressed on 5 bits ([4:0] bits).

Parameters

- **LPUARTx:** LPUART Instance
- **Time:** Value between Min_Data=0 and Max_Data=31

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_SetDEAssertionTime

LL_LPUART_GetDEAssertionTime

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDEAssertionTime (const USART_TypeDef * LPUARTx)

Function description

Return DEAT (Driver Enable Assertion Time)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** value expressed on 5 bits ([4:0] bits) : Time Value between Min_Data=0 and Max_Data=31

Reference Manual to LL API cross reference:

- CR1 DEAT LL_LPUART_GetDEAssertionTime

LL_LPUART_EnableDEMode

Function name

__STATIC_INLINE void LL_LPUART_EnableDEMode (USART_TypeDef * LPUARTx)

Function description

Enable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_EnableDEMode

LL_LPUART_DisableDEMode

Function name

__STATIC_INLINE void LL_LPUART_DisableDEMode (USART_TypeDef * LPUARTx)

Function description

Disable Driver Enable (DE) Mode.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_DisableDEMode

LL_LPUART_IsEnabledDEMode

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDEMode (const USART_TypeDef * LPUARTx)

Function description

Indicate if Driver Enable (DE) Mode is enabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 DEM LL_LPUART_IsEnabledDEMode

LL_LPUART_SetDESignalPolarity

Function name

__STATIC_INLINE void LL_LPUART_SetDESignalPolarity (USART_TypeDef * LPUARTx, uint32_t Polarity)

Function description

Select Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance
- **Polarity:** This parameter can be one of the following values:
  - LL_LPUART_DE_POLARITY_HIGH
  - LL_LPUART_DE_POLARITY_LOW

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_SetDESignalPolarity

LL_LPUART_GetDESignalPolarity

Function name

__STATIC_INLINE uint32_t LL_LPUART_GetDESignalPolarity (const USART_TypeDef * LPUARTx)

Function description

Return Driver Enable Polarity.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_LPUART_DE_POLARITY_HIGH
  – LL_LPUART_DE_POLARITY_LOW

Reference Manual to LL API cross reference:

- CR3 DEP LL_LPUART_GetDESignalPolarity

LL_LPUART_IsActiveFlag_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR PE LL_LPUART_IsActiveFlag_PE

LL_LPUART_IsActiveFlag_FE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_FE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Framing Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR FE LL_LPUART_IsActiveFlag_FE

LL_LPUART_IsActiveFlag_NE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_NE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Noise error detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR NE LL_LPUART_IsActiveFlag_NE

LL_LPUART_IsActiveFlag_ORE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_ORE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART OverRun Error Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR ORE LL_LPUART_IsActiveFlag_ORE

LL_LPUART_IsActiveFlag_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE line detected Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR IDLE LL_LPUART_IsActiveFlag_IDLE

LL_LPUART_IsActiveFlag_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Read Data Register or LPUART RX FIFO Not Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR RXNE_RXFNE LL_LPUART_IsActiveFlag_RXNE_RXFNE

LL_LPUART_IsActiveFlag_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TC LL_LPUART_IsActiveFlag_TC

LL_LPUART_IsActiveFlag_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Data Register Empty or LPUART TX FIFO Not Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TXE_TXFNF LL_LPUART_IsActiveFlag_TXE_TXFNF

LL_LPUART_IsActiveFlag_nCTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_nCTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS interrupt Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR CTSIF LL_LPUART_IsActiveFlag_nCTS

LL_LPUART_IsActiveFlag_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR CTS LL_LPUART_IsActiveFlag_CTS

LL_LPUART_IsActiveFlag_BUSY

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_BUSY (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Busy Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR BUSY LL_LPUART_IsActiveFlag_BUSY

LL_LPUART_IsActiveFlag_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR CMF LL_LPUART_IsActiveFlag_CM

LL_LPUART_IsActiveFlag_SBK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_SBK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Send Break Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR SBKF LL_LPUART_IsActiveFlag_SBK

LL_LPUART_IsActiveFlag_RWU

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RWU (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Wake Up from mute mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR RWU LL_LPUART_IsActiveFlag_RWU

LL_LPUART_IsActiveFlag_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from stop mode Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR WUF LL_LPUART_IsActiveFlag_WKUP

LL_LPUART_IsActiveFlag_TEACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TEACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmit Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TEACK LL_LPUART_IsActiveFlag_TEACK

LL_LPUART_IsActiveFlag_REACK

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_REACK (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Receive Enable Acknowledge Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR REACK LL_LPUART_IsActiveFlag_REACK

LL_LPUART_IsActiveFlag_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TXFE LL_LPUART_IsActiveFlag_TXFE

LL_LPUART_IsActiveFlag_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR RXFF LL_LPUART_IsActiveFlag_RXFF

LL_LPUART_IsActiveFlag_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR TXFT LL_LPUART_IsActiveFlag_TXFT

LL_LPUART_IsActiveFlag_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsActiveFlag_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Threshold Flag is set or not.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR RXFT LL_LPUART_IsActiveFlag_RXFT

LL_LPUART_ClearFlag_PE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_PE (USART_TypeDef * LPUARTx)

Function description

Clear Parity Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR PECF LL_LPUART_ClearFlag_PE

LL_LPUART_ClearFlag_FE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_FE (USART_TypeDef * LPUARTx)

Function description

Clear Framing Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR FECF LL_LPUART_ClearFlag_FE

LL_LPUART_ClearFlag_NE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_NE (USART_TypeDef * LPUARTx)

Function description

Clear Noise detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR NECF LL_LPUART_ClearFlag_NE

LL_LPUART_ClearFlag_ORE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_ORE (USART_TypeDef * LPUARTx)

Function description

Clear OverRun Error Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR ORECF LL_LPUART_ClearFlag_ORE

LL_LPUART_ClearFlag_IDLE

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_IDLE (USART_TypeDef * LPUARTx)

Function description

Clear IDLE line detected Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR IDLECF LL_LPUART_ClearFlag_IDLE

LL_LPUART_ClearFlag_TC

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_TC (USART_TypeDef * LPUARTx)

Function description

Clear Transmission Complete Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR TCCF LL_LPUART_ClearFlag_TC

LL_LPUART_ClearFlag_nCTS

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_nCTS (USART_TypeDef * LPUARTx)

Function description

Clear CTS Interrupt Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CTSCF LL_LPUART_ClearFlag_nCTS

LL_LPUART_ClearFlag_CM

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_CM (USART_TypeDef * LPUARTx)

Function description

Clear Character Match Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR CMCF LL_LPUART_ClearFlag_CM

LL_LPUART_ClearFlag_WKUP

Function name

__STATIC_INLINE void LL_LPUART_ClearFlag_WKUP (USART_TypeDef * LPUARTx)

Function description

Clear Wake Up from stop mode Flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICR WUCF LL_LPUART_ClearFlag_WKUP

LL_LPUART_EnableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Enable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_EnableIT_IDLE

LL_LPUART_EnableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Enable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_EnableIT_RXNE_RXFNE

LL_LPUART_EnableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TC (USART_TypeDef * LPUARTx)

Function description

Enable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_EnableIT_TC

LL_LPUART_EnableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Enable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_EnableIT_TXE_TXFNF

LL_LPUART_EnableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_PE (USART_TypeDef * LPUARTx)

Function description

Enable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_EnableIT_PE

LL_LPUART_EnableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CM (USART_TypeDef * LPUARTx)

Function description

Enable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_EnableIT_CM

LL_LPUART_EnableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_EnableIT_TXFE

LL_LPUART_EnableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_EnableIT_RXFF

LL_LPUART_EnableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Enable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_EnableIT_ERROR

LL_LPUART_EnableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Enable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_EnableIT_CTS

LL_LPUART_EnableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Enable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_EnableIT_WKUP

LL_LPUART_EnableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Enable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_EnableIT_TXFT

LL_LPUART_EnableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_EnableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Enable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_EnableIT_RXFT

LL_LPUART_DisableIT_IDLE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_IDLE (USART_TypeDef * LPUARTx)

Function description

Disable IDLE Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_DisableIT_IDLE

LL_LPUART_DisableIT_RXNE_RXFNE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXNE_RXFNE (USART_TypeDef * LPUARTx)

Function description

Disable RX Not Empty and RX FIFO Not Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_DisableIT_RXNE_RXFNE

LL_LPUART_DisableIT_TC

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TC (USART_TypeDef * LPUARTx)

Function description

Disable Transmission Complete Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_DisableIT_TC

LL_LPUART_DisableIT_TXE_TXFNF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXE_TXFNF (USART_TypeDef * LPUARTx)

Function description

Disable TX Empty and TX FIFO Not Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_DisableIT_TXE_TXFNF

LL_LPUART_DisableIT_PE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_PE (USART_TypeDef * LPUARTx)

Function description

Disable Parity Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_DisableIT_PE

LL_LPUART_DisableIT_CM

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CM (USART_TypeDef * LPUARTx)

Function description

Disable Character Match Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_DisableIT_CM

LL_LPUART_DisableIT_TXFE

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFE (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Empty Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_DisableIT_TXFE

LL_LPUART_DisableIT_RXFF

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFF (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Full Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_DisableIT_RXFF

LL_LPUART_DisableIT_ERROR

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_ERROR (USART_TypeDef * LPUARTx)

Function description

Disable Error Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- When set, Error Interrupt Enable Bit is enabling interrupt generation in case of a framing error, overrun error or noise flag (FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register). 0: Interrupt is inhibited1: An interrupt is generated when FE=1 or ORE=1 or NF=1 in the LPUARTx_ISR register.

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_DisableIT_ERROR

LL_LPUART_DisableIT_CTS

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_CTS (USART_TypeDef * LPUARTx)

Function description

Disable CTS Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_DisableIT_CTS

LL_LPUART_DisableIT_WKUP

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_WKUP (USART_TypeDef * LPUARTx)

Function description

Disable Wake Up from Stop Mode Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_DisableIT_WKUP

LL_LPUART_DisableIT_TXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_TXFT (USART_TypeDef * LPUARTx)

Function description

Disable TX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_DisableIT_TXFT

LL_LPUART_DisableIT_RXFT

Function name

__STATIC_INLINE void LL_LPUART_DisableIT_RXFT (USART_TypeDef * LPUARTx)

Function description

Disable RX FIFO Threshold Interrupt.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_DisableIT_RXFT

LL_LPUART_IsEnabledIT_IDLE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_IDLE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART IDLE Interrupt source is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 IDLEIE LL_LPUART_IsEnabledIT_IDLE

LL_LPUART_IsEnabledIT_RXNE_RXFNE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXNE_RXFNE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX Not Empty and LPUART RX FIFO Not Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 RXNEIE_RXFNEIE LL_LPUART_IsEnabledIT_RXNE_RXFNE

LL_LPUART_IsEnabledIT_TC

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TC (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Transmission Complete Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 TCIE LL_LPUART_IsEnabledIT_TC

LL_LPUART_IsEnabledIT_TXE_TXFNF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXE_TXFNF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX Empty and LPUART TX FIFO Not Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 TXEIE_TXFNFIE LL_LPUART_IsEnabledIT_TXE_TXFNF

LL_LPUART_IsEnabledIT_PE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_PE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Parity Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 PEIE LL_LPUART_IsEnabledIT_PE

LL_LPUART_IsEnabledIT_CM

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CM (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Character Match Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 CMIE LL_LPUART_IsEnabledIT_CM

LL_LPUART_IsEnabledIT_TXFE

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFE (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART TX FIFO Empty Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 TXFEIE LL_LPUART_IsEnabledIT_TXFE

LL_LPUART_IsEnabledIT_RXFF

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFF (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART RX FIFO Full Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR1 RXFFIE LL_LPUART_IsEnabledIT_RXFF

LL_LPUART_IsEnabledIT_ERROR

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_ERROR (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Error Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 EIE LL_LPUART_IsEnabledIT_ERROR

LL_LPUART_IsEnabledIT_CTS

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_CTS (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART CTS Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 CTSIE LL_LPUART_IsEnabledIT_CTS

LL_LPUART_IsEnabledIT_WKUP

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_WKUP (const USART_TypeDef * LPUARTx)

Function description

Check if the LPUART Wake Up from Stop Mode Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 WUFIE LL_LPUART_IsEnabledIT_WKUP

LL_LPUART_IsEnabledIT_TXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_TXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART TX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 TXFTIE LL_LPUART_IsEnabledIT_TXFT

LL_LPUART_IsEnabledIT_RXFT

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledIT_RXFT (const USART_TypeDef * LPUARTx)

Function description

Check if LPUART RX FIFO Threshold Interrupt is enabled or disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 RXFTIE LL_LPUART_IsEnabledIT_RXFT

LL_LPUART_EnableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_EnableDMAReq_RX

LL_LPUART_DisableDMAReq_RX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_RX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_DisableDMAReq_RX

LL_LPUART_IsEnabledDMAReq_RX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_RX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for reception.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 DMAR LL_LPUART_IsEnabledDMAReq_RX

LL_LPUART_EnableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_EnableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Enable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_EnableDMAReq_TX

LL_LPUART_DisableDMAReq_TX

Function name

__STATIC_INLINE void LL_LPUART_DisableDMAReq_TX (USART_TypeDef * LPUARTx)

Function description

Disable DMA Mode for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_DisableDMAReq_TX

LL_LPUART_IsEnabledDMAReq_TX

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMAReq_TX (const USART_TypeDef * LPUARTx)

Function description

Check if DMA Mode is enabled for transmission.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 DMAT LL_LPUART_IsEnabledDMAReq_TX

LL_LPUART_EnableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_EnableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Enable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_EnableDMADeactOnRxErr

LL_LPUART_DisableDMADeactOnRxErr

Function name

__STATIC_INLINE void LL_LPUART_DisableDMADeactOnRxErr (USART_TypeDef * LPUARTx)

Function description

Disable DMA Disabling on Reception Error.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_DisableDMADeactOnRxErr

LL_LPUART_IsEnabledDMADeactOnRxErr

Function name

__STATIC_INLINE uint32_t LL_LPUART_IsEnabledDMADeactOnRxErr (const USART_TypeDef * LPUARTx)

Function description

Indicate if DMA Disabling on Reception Error is disabled.

Parameters

- **LPUARTx:** LPUART Instance

Return values

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

Reference Manual to LL API cross reference:

- CR3 DDRE LL_LPUART_IsEnabledDMADeactOnRxErr

LL_LPUART_DMA_GetRegAddr

Function name

__STATIC_INLINE uint32_t LL_LPUART_DMA_GetRegAddr (const USART_TypeDef * LPUARTx, uint32_t Direction)

Function description

Get the LPUART data register address used for DMA transfer.

Parameters

- **LPUARTx:** LPUART Instance
- **Direction:** This parameter can be one of the following values:
  - LL_LPUART_DMA_REG_DATA_TRANSMIT
  - LL_LPUART_DMA_REG_DATA_RECEIVE

Return values

- **Address:** of data register

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_DMA_GetRegAddr
-
- TDR TDR LL_LPUART_DMA_GetRegAddr

LL_LPUART_ReceiveData8

Function name

__STATIC_INLINE uint8_t LL_LPUART_ReceiveData8 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0xFF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData8

LL_LPUART_ReceiveData9

Function name

__STATIC_INLINE uint16_t LL_LPUART_ReceiveData9 (const USART_TypeDef * LPUARTx)

Function description

Read Receiver Data register (Receive Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **Time:** Value between Min_Data=0x00 and Max_Data=0x1FF

Reference Manual to LL API cross reference:

- RDR RDR LL_LPUART_ReceiveData9

LL_LPUART_TransmitData8

Function name

__STATIC_INLINE void LL_LPUART_TransmitData8 (USART_TypeDef * LPUARTx, uint8_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 8 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0xFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData8

LL_LPUART_TransmitData9

Function name

__STATIC_INLINE void LL_LPUART_TransmitData9 (USART_TypeDef * LPUARTx, uint16_t Value)

Function description

Write in Transmitter Data Register (Transmit Data value, 9 bits)

Parameters

- **LPUARTx:** LPUART Instance
- **Value:** between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Reference Manual to LL API cross reference:

- TDR TDR LL_LPUART_TransmitData9

LL_LPUART_RequestBreakSending

Function name

__STATIC_INLINE void LL_LPUART_RequestBreakSending (USART_TypeDef * LPUARTx)

Function description

Request Break sending.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR SBKRQ LL_LPUART_RequestBreakSending

LL_LPUART_RequestEnterMuteMode

Function name

__STATIC_INLINE void LL_LPUART_RequestEnterMuteMode (USART_TypeDef * LPUARTx)

Function description

Put LPUART in mute mode and set the RWU flag.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- RQR MMRQ LL_LPUART_RequestEnterMuteMode

LL_LPUART_RequestRxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestRxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Receive Data and FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- Allows to discard the received data without reading them, and avoid an overrun condition.

Reference Manual to LL API cross reference:

- RQR RXFRQ LL_LPUART_RequestRxDataFlush

LL_LPUART_RequestTxDataFlush

Function name

__STATIC_INLINE void LL_LPUART_RequestTxDataFlush (USART_TypeDef * LPUARTx)

Function description

Request a Transmit data FIFO flush.

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **None:**

Notes

- TXFRQ bit is set to flush the whole FIFO when FIFO mode is enabled. This also sets the flag TXFE (TXFIFO empty bit in the LPUART_ISR register).
- Macro IS_UART_FIFO_INSTANCE(USARTx) can be used to check whether or not FIFO mode feature is supported by the USARTx instance.

Reference Manual to LL API cross reference:

- RQR TXFRQ LL_LPUART_RequestTxDataFlush

LL_LPUART_DeInit

Function name

ErrorStatus LL_LPUART_DeInit (const USART_TypeDef * LPUARTx)

Function description

De-initialize LPUART registers (Registers restored to their default values).

Parameters

- **LPUARTx:** LPUART Instance

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are de-initialized
  - ERROR: not applicable

LL_LPUART_Init

Function name

ErrorStatus LL_LPUART_Init (USART_TypeDef * LPUARTx, const LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Initialize LPUART registers according to the specified parameters in LPUART_InitStruct.

Parameters

- **LPUARTx:** LPUART Instance
- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure that contains the configuration information for the specified LPUART peripheral.

Return values

- **An:** ErrorStatus enumeration value:
  - SUCCESS: LPUART registers are initialized according to LPUART_InitStruct content
  - ERROR: Problem occurred during LPUART Registers initialization

Notes

- As some bits in LPUART configuration registers can only be written when the LPUART is disabled (USART_CR1_UE bit =0), LPUART Peripheral should be in disabled state prior calling this function. Otherwise, ERROR result will be returned.
- Baud rate value stored in LPUART_InitStruct BaudRate field, should be valid (different from 0).

LL_LPUART_StructInit

Function name

void LL_LPUART_StructInit (LL_LPUART_InitTypeDef * LPUART_InitStruct)

Function description

Set each LL_LPUART_InitTypeDef field to default value.

Parameters

- **LPUART_InitStruct:** pointer to a LL_LPUART_InitTypeDef structure whose fields will be set to default values.

Return values

- **None:**

54.3 LPUART Firmware driver defines

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

54.3.1 LPUART

LPUART

Address Length Detection

LL_LPUART_ADDRESS_DETECT_4B

4-bit address detection method selected

LL_LPUART_ADDRESS_DETECT_7B

7-bit address detection (in 8-bit data mode) method selected

Binary Data Inversion

LL_LPUART_BINARY_LOGIC_POSITIVE

Logical data from the data register are send/received in positive/direct logic. (1=H, 0=L)

LL_LPUART_BINARY_LOGIC_NEGATIVE

Logical data from the data register are send/received in negative/inverse logic. (1=L, 0=H). The parity bit is also inverted.

Bit Order

LL_LPUART_BITORDER_LSBFIRST

data is transmitted/received with data bit 0 first, following the start bit

LL_LPUART_BITORDER_MSBFIRST

data is transmitted/received with the MSB first, following the start bit

Clear Flags Defines

LL_LPUART_ICR_PECF

Parity error clear flag

LL_LPUART_ICR_FECF

Framing error clear flag

LL_LPUART_ICR_NCF

Noise error detected clear flag

LL_LPUART_ICR_ORECF

Overrun error clear flag

LL_LPUART_ICR_IDLECF

Idle line detected clear flag

LL_LPUART_ICR_TCCF

Transmission complete clear flag

LL_LPUART_ICR_CTSCF

CTS clear flag

LL_LPUART_ICR_CMCF

Character match clear flag

LL_LPUART_ICR_WUCF

Wakeup from Stop mode clear flag

Datawidth

LL_LPUART_DATAWIDTH_7B

7 bits word length : Start bit, 7 data bits, n stop bits

LL_LPUART_DATAWIDTH_8B

8 bits word length : Start bit, 8 data bits, n stop bits

LL_LPUART_DATAWIDTH_9B

9 bits word length : Start bit, 9 data bits, n stop bits

Driver Enable Polarity

LL_LPUART_DE_POLARITY_HIGH

DE signal is active high

LL_LPUART_DE_POLARITY_LOW

DE signal is active low

Direction

LL_LPUART_DIRECTION_NONE

Transmitter and Receiver are disabled

LL_LPUART_DIRECTION_RX

Transmitter is disabled and Receiver is enabled

LL_LPUART_DIRECTION_TX

Transmitter is enabled and Receiver is disabled

LL_LPUART_DIRECTION_TX_RX

Transmitter and Receiver are enabled

DMA Register Data

LL_LPUART_DMA_REG_DATA_TRANSMIT

Get address of data register used for transmission

LL_LPUART_DMA_REG_DATA_RECEIVE

Get address of data register used for reception

FIFO Threshold

LL_LPUART_FIFOTHRESHOLD_1_8

FIFO reaches 1/8 of its depth

LL_LPUART_FIFOTHRESHOLD_1_4

FIFO reaches 1/4 of its depth

LL_LPUART_FIFOTHRESHOLD_1_2

FIFO reaches 1/2 of its depth

LL_LPUART_FIFOTHRESHOLD_3_4

FIFO reaches 3/4 of its depth

LL_LPUART_FIFOTHRESHOLD_7_8

FIFO reaches 7/8 of its depth

LL_LPUART_FIFOTHRESHOLD_8_8

FIFO becomes empty for TX and full for RX

Get Flags Defines

LL_LPUART_ISR_PE

Parity error flag

LL_LPUART_ISR_FE

Framing error flag

LL_LPUART_ISR_NE

Noise detected flag

LL_LPUART_ISR_ORE

Overrun error flag

LL_LPUART_ISR_IDLE

Idle line detected flag

LL_LPUART_ISR_RXNE_RXFNE

Read data register or RX FIFO not empty flag

LL_LPUART_ISR_TC

Transmission complete flag

LL_LPUART_ISR_TXE_TXFNF

Transmit data register empty or TX FIFO Not Full flag

LL_LPUART_ISR_CTSIF

CTS interrupt flag

LL_LPUART_ISR_CTS

CTS flag

LL_LPUART_ISR_BUSY

Busy flag

LL_LPUART_ISR_CMF

Character match flag

LL_LPUART_ISR_SBKF

Send break flag

LL_LPUART_ISR_RWU

Receiver wakeup from Mute mode flag

LL_LPUART_ISR_WUF

Wakeup from Stop mode flag

LL_LPUART_ISR_TEACK

Transmit enable acknowledge flag

LL_LPUART_ISR_REACK

Receive enable acknowledge flag

LL_LPUART_ISR_TXFE

TX FIFO empty flag

LL_LPUART_ISR_RXFF

RX FIFO full flag

LL_LPUART_ISR_RXFT

RX FIFO threshold flag

LL_LPUART_ISR_TXFT

TX FIFO threshold flag

Hardware Control

LL_LPUART_HWCONTROL_NONE

CTS and RTS hardware flow control disabled

LL_LPUART_HWCONTROL_RTS

RTS output enabled, data is only requested when there is space in the receive buffer

LL_LPUART_HWCONTROL_CTS

CTS mode enabled, data is only transmitted when the nCTS input is asserted (tied to 0)

LL_LPUART_HWCONTROL_RTS_CTS

CTS and RTS hardware flow control enabled

IT Defines

LL_LPUART_CR1_IDLEIE

IDLE interrupt enable

LL_LPUART_CR1_RXNEIE_RXFNEIE

Read data register and RXFIFO not empty interrupt enable

LL_LPUART_CR1_TCIE

Transmission complete interrupt enable

LL_LPUART_CR1_TXEIE_TXFNFIE

Transmit data register empty and TX FIFO not full interrupt enable

LL_LPUART_CR1_PEIE

Parity error

LL_LPUART_CR1_CMIE

Character match interrupt enable

LL_LPUART_CR1_TXFEIE

TX FIFO empty interrupt enable

LL_LPUART_CR1_RXFFIE

RX FIFO full interrupt enable

LL_LPUART_CR3_EIE

Error interrupt enable

LL_LPUART_CR3_CTSIE

CTS interrupt enable

LL_LPUART_CR3_WUFIE

Wakeup from Stop mode interrupt enable

LL_LPUART_CR3_TXFTIE

TX FIFO threshold interrupt enable

LL_LPUART_CR3_RXFTIE

RX FIFO threshold interrupt enable

Parity Control

LL_LPUART_PARITY_NONE

Parity control disabled

LL_LPUART_PARITY_EVEN

Parity control enabled and Even Parity is selected

LL_LPUART_PARITY_ODD

Parity control enabled and Odd Parity is selected

Clock Source Prescaler

LL_LPUART_PRESCALER_DIV1

Input clock not divided

LL_LPUART_PRESCALER_DIV2

Input clock divided by 2

LL_LPUART_PRESCALER_DIV4

Input clock divided by 4

LL_LPUART_PRESCALER_DIV6

Input clock divided by 6

LL_LPUART_PRESCALER_DIV8

Input clock divided by 8

LL_LPUART_PRESCALER_DIV10

Input clock divided by 10

LL_LPUART_PRESCALER_DIV12

Input clock divided by 12

LL_LPUART_PRESCALER_DIV16

Input clock divided by 16

LL_LPUART_PRESCALER_DIV32

Input clock divided by 32

LL_LPUART_PRESCALER_DIV64

Input clock divided by 64

LL_LPUART_PRESCALER_DIV128

Input clock divided by 128

LL_LPUART_PRESCALER_DIV256

Input clock divided by 256

RX Pin Active Level Inversion

LL_LPUART_RXPIN_LEVEL_STANDARD

RX pin signal works using the standard logic levels

LL_LPUART_RXPIN_LEVEL_INVERTED

RX pin signal values are inverted.

Stop Bits

LL_LPUART_STOPBITS_1

1 stop bit

LL_LPUART_STOPBITS_2

2 stop bits

TX Pin Active Level Inversion

LL_LPUART_TXPIN_LEVEL_STANDARD

TX pin signal works using the standard logic levels

LL_LPUART_TXPIN_LEVEL_INVERTED

TX pin signal values are inverted.

TX RX Pins Swap

LL_LPUART_TXRX_STANDARD

TX/RX pins are used as defined in standard pinout

LL_LPUART_TXRX_SWAPPED

TX and RX pins functions are swapped.

Wakeup

LL_LPUART_WAKEUP_IDLELINE

LPUART wake up from Mute mode on Idle Line

LL_LPUART_WAKEUP_ADDRESSMARK

LPUART wake up from Mute mode on Address Mark

Wakeup Activation

LL_LPUART_WAKEUP_ON_ADDRESS

Wake up active on address match

LL_LPUART_WAKEUP_ON_STARTBIT

Wake up active on Start bit detection

LL_LPUART_WAKEUP_ON_RXNE

Wake up active on RXNE

FLAG_Management

LL_LPUART_IsActiveFlag_RXNE

LL_LPUART_IsActiveFlag_TXE

IT_Management

LL_LPUART_EnableIT_RXNE

LL_LPUART_EnableIT_TXE

LL_LPUART_DisableIT_RXNE

LL_LPUART_DisableIT_TXE

LL_LPUART_IsEnabledIT_RXNE

LL_LPUART_IsEnabledIT_TXE

Helper Macros

__LL_LPUART_DIV

Description:

- Compute LPUARTDIV value according to Peripheral Clock and expected Baud Rate (20-bit value of LPUARTDIV is returned)

Parameters:

- __PERIPHCLK__: Peripheral Clock frequency used for LPUART Instance
- __PRESCALER__: This parameter can be one of the following values:
  - LL_LPUART_PRESCALER_DIV1
  - LL_LPUART_PRESCALER_DIV2
  - LL_LPUART_PRESCALER_DIV4
  - LL_LPUART_PRESCALER_DIV6
  - LL_LPUART_PRESCALER_DIV8
  - LL_LPUART_PRESCALER_DIV10
  - LL_LPUART_PRESCALER_DIV12
  - LL_LPUART_PRESCALER_DIV16
  - LL_LPUART_PRESCALER_DIV32
  - LL_LPUART_PRESCALER_DIV64
  - LL_LPUART_PRESCALER_DIV128
  - LL_LPUART_PRESCALER_DIV256
- __BAUDRATE__: Baud Rate value to achieve

Return value:

- LPUARTDIV: value to be used for BRR register filling

Common Write and read registers Macros

LL_LPUART_WriteReg

Description:

- Write a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_LPUART_ReadReg

Description:

- Read a value in LPUART register.

Parameters:

- __INSTANCE__: LPUART Instance
- __REG__: Register to be read

Return value:

- Register: value

55 LL PKA Generic Driver

55.1 PKA Firmware driver API description

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

55.1.1 Detailed description of functions

LL_PKA_DeInit

Function name

ErrorStatus LL_PKA_DeInit (const PKA_TypeDef * PKAx)

Function description

De-initialize PKA registers (Registers restored to their default values).

Parameters

- **PKAx:** PKA Instance.

Return values

- **ErrorStatus:**
  – SUCCESS: PKA registers are de-initialized
  – ERROR: PKA registers are not de-initialized

55.2 PKA Firmware driver defines

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

55.2.1 PKA

PKA

IT Defines

LL_PKA_CR_ADDRERRIE

LL_PKA_CR_RAMERRIE

LL_PKA_CR_PROCENDIE

LL_PKA_CLRFR_PROCENDFC

LL_PKA_CLRFR_RAMERRFC

LL_PKA_CLRFR_ADDRERRFC

Common Write and read registers Macros

LL_PKA_WriteReg

Description:

- Write a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:

- None

LL_PKA_ReadReg

Description:

- Read a value in PKA register.

Parameters:

- __INSTANCE__: PKA Instance
- __REG__: Register to be read

Return value:

- Register: value

56 LL PWR Generic Driver

56.1 PWR Firmware driver API description

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

56.1.1 Detailed description of functions

LL_PWR_EnableBORinSDN

Function name

__STATIC_INLINE void LL_PWR_EnableBORinSDN (void)

Function description

Enable the BOR reset supervising during SHUTDOWN mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ENSDNBOR LL_PWR_EnableBORinSDN

LL_PWR_DisableBORinSDN

Function name

__STATIC_INLINE void LL_PWR_DisableBORinSDN (void)

Function description

Disable the BOR reset supervising during SHUTDOWN mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR1 ENSDNBOR LL_PWR_DisableBORinSDN

LL_PWR_IsEnabledBORinSDN

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledBORinSDN (void)

Function description

Checks if BOR is enabled or disabled during SHUTDOWN mode.

Return values

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

Reference Manual to LL API cross reference:

- CR1 ENSDNBOR LL_PWR_IsEnabledBORinSDN

LL_PWR_SetPowerMode

Function name

__STATIC_INLINE void LL_PWR_SetPowerMode (uint32_t LowPowerMode)

Function description

Set low-power mode.

Parameters

- **LowPowerMode:** Low Power Mode Selection. This parameter can be one of the following values:
  - LL_PWR_MODE_DEEPSTOP
  - 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_DEEPSTOP
  - LL_PWR_MODE_SHUTDOWN

Reference Manual to LL API cross reference:

- CR1 LPMS LL_PWR_GetPowerMode

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:

- CR1 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:

- CR1 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:

- CR1 APC LL_PWR_IsEnabledPUPDCfg

LL_PWR_EnableRAMBankRet

Function name

__STATIC_INLINE void LL_PWR_EnableRAMBankRet (uint32_t banks)

Function description

Enable the RAM bank retention.

Parameters

- **banks:** RAM bank selection. This parameter can be a combination of the following values:
  – LL_PWR_RAMRET_1
  – LL_PWR_RAMRET_2 (*)
  – LL_PWR_RAMRET_3 (*) (*) Not available on devices STM32WB05

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RAMRET1 LL_PWR_DisableRAMBankRet CR2 RAMRET2 LL_PWR_DisableRAMBankRet CR2 RAMRET3 LL_PWR_DisableRAMBankRet

LL_PWR_GetRAMBankRet

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRAMBankRet (void)

Function description

Get the RAM banks retention selected.

Return values

- **Returned:** value can be one of the following values:
  – LL_PWR_RAMRET_1
  – LL_PWR_RAMRET_2 (*)
  – LL_PWR_RAMRET_3 (*) (*) Not available on devices STM32WB05

Reference Manual to LL API cross reference:

- CR2 RAMRET1 LL_PWR_DisableRAMBankRet CR2 RAMRET2 LL_PWR_DisableRAMBankRet CR2 RAMRET3 LL_PWR_DisableRAMBankRet

LL_PWR_DisableRAMBankRet

Function name

__STATIC_INLINE void LL_PWR_DisableRAMBankRet (uint32_t banks)

Function description

Disable the RAM bank retention.

Parameters

- **banks:** RAM bank selection. This parameter can be a combination of the following values:
  – LL_PWR_RAMRET_1
  – LL_PWR_RAMRET_2 (*)
  – LL_PWR_RAMRET_3 (*) (*) Not available on devices STM32WB05

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 RAMRET1 LL_PWR_DisableRAMBankRet CR2 RAMRET2 LL_PWR_DisableRAMBankRet CR2 RAMRET3 LL_PWR_DisableRAMBankRet

LL_PWR_SetPVDLevel

Function name

__STATIC_INLINE void LL_PWR_SetPVDLevel (uint32_t PVDLevel)

Function description

Configure the voltage threshold detected by the Power Voltage Detector.

Parameters

- **PVDLevel:** PVD voltage level selection. This parameter can be one of the following values:
  – LL_PWR_PVDLEVEL_0
  – LL_PWR_PVDLEVEL_1
  – LL_PWR_PVDLEVEL_2
  – LL_PWR_PVDLEVEL_3
  – LL_PWR_PVDLEVEL_4
  – LL_PWR_PVDLEVEL_5
  – LL_PWR_PVDLEVEL_6
  – LL_PWR_PVDLEVEL_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVDLS LL_PWR_SetPVDLevel

LL_PWR_GetPVDLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_GetPVDLevel (void)

Function description

Get the voltage threshold detection.

Return values

- **Returned:** value can be one of the following values:
  - LL_PWR_PVDLEVEL_0
  - LL_PWR_PVDLEVEL_1
  - LL_PWR_PVDLEVEL_2
  - LL_PWR_PVDLEVEL_3
  - LL_PWR_PVDLEVEL_4
  - LL_PWR_PVDLEVEL_5
  - LL_PWR_PVDLEVEL_6
  - LL_PWR_PVDLEVEL_7

Reference Manual to LL API cross reference:

- CR2 PVDLS LL_PWR_GetPVDLevel

LL_PWR_EnablePVD

Function name

__STATIC_INLINE void LL_PWR_EnablePVD (void)

Function description

Enable Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVDE LL_PWR_EnablePVD

LL_PWR_DisablePVD

Function name

__STATIC_INLINE void LL_PWR_DisablePVD (void)

Function description

Disable the Power Voltage Detector.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 PVDE LL_PWR_DisablePVD

LL_PWR_IsEnabledPVD

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledPVD (void)

Function description

Check if Power Voltage Detector is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CR2 PVDE LL_PWR_IsEnabledPVD

LL_PWR_EnableGPIORET

Function name

__STATIC_INLINE void LL_PWR_EnableGPIORET (void)

Function description

Enable the GPIORET feature, GPIO retain their status during DEEPSTOP and exiting from DEEPSTOP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 GPIORET LL_PWR_EnableGPIORET

LL_PWR_DisableGPIORET

Function name

__STATIC_INLINE void LL_PWR_DisableGPIORET (void)

Function description

Disable the GPIORET feature, don't retain their status during DEEPSTOP and exiting from DEEPSTOP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 GPIORET LL_PWR_DisableGPIORET

LL_PWR_IsEnabledGPIORET

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIORET (void)

Function description

Checks if the GPIO retention in DEEPSTOP feature is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CR2 GPIORET LL_PWR_IsEnabledGPIORET

LL_PWR_EnableDBGRET

Function name

__STATIC_INLINE void LL_PWR_EnableDBGRET (void)

Function description

Enable the DBGRET feature, GPIOA_Pin_2 and GPIOA_Pin_3 retain their status during DEEPSTOP and exiting from DEEPSTOP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DBGRET LL_PWR_EnableDBGRET

LL_PWR_DisableDBGRET

Function name

__STATIC_INLINE void LL_PWR_DisableDBGRET (void)

Function description

Disable the DBGRET feature, GPIOA_Pin_2 and GPIOA_Pin_3 don't retain status during DEEPSTOP and exiting from DEEPSTOP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR2 DBGRET LL_PWR_DisableDBGRET

LL_PWR_IsEnabledDBGRET

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledDBGRET (void)

Function description

Checks if the GPIOA_Pin_2 and GPIOA_Pin_3 retention in DEEPSTOP feature is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CR2 DBGRET LL_PWR_IsEnabledDBGRET

LL_PWR_EnableInternWU

Function name

__STATIC_INLINE void LL_PWR_EnableInternWU (void)

Function description

Enable Internal Wake-up line (RTC)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWL LL_PWR_EnableInternWU

LL_PWR_DisableInternWU

Function name

__STATIC_INLINE void LL_PWR_DisableInternWU (void)

Function description

Disable Internal Wake-up line (RTC)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWL 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 (RTC)

Return values

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

Reference Manual to LL API cross reference:

- CR3 EIWL LL_PWR_IsEnabledInternWU

LL_PWR_EnableInternWU2

Function name

__STATIC_INLINE void LL_PWR_EnableInternWU2 (void)

Function description

Enable Internal Wake-up line (LPUART)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWL2 LL_PWR_EnableInternWU2

LL_PWR_DisableInternWU2

Function name

__STATIC_INLINE void LL_PWR_DisableInternWU2 (void)

Function description

Disable Internal Wake-up line (LPUART)

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EIWL2 LL_PWR_EnableInternWU2

LL_PWR_IsEnabledInternWU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledInternWU2 (void)

Function description

Check if Internal Wake-up line is enabled (LPUART)

Return values

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

Reference Manual to LL API cross reference:

- CR3 EIWL2 LL_PWR_EnableInternWU2

LL_PWR_ClearFlag_InternWU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_InternWU2 (void)

Function description

Clear Internal Wake-up2 flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 IWUF2 LL_PWR_ClearFlag_BORH

LL_PWR_EnableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_EnableWakeUpPin (uint32_t WakeUpPin)

Function description

Enable Wakeup PINx to get out of DEEPSTOP mode.

Parameters

- **WakeUpPin:** Enable the wakeup sources. This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PA0
  - LL_PWR_WAKEUP_PA1
  - LL_PWR_WAKEUP_PA2
  - LL_PWR_WAKEUP_PA3
  - LL_PWR_WAKEUP_PA4 (*)
  - LL_PWR_WAKEUP_PA5 (*)
  - LL_PWR_WAKEUP_PA6 (*)
  - LL_PWR_WAKEUP_PA7 (*)
  - LL_PWR_WAKEUP_PA8
  - LL_PWR_WAKEUP_PA9
  - LL_PWR_WAKEUP_PA10
  - LL_PWR_WAKEUP_PA11
  - LL_PWR_WAKEUP_PA12 (*)
  - LL_PWR_WAKEUP_PA13 (*)
  - LL_PWR_WAKEUP_PA14 (*)
  - LL_PWR_WAKEUP_PA15 (*)
  - LL_PWR_WAKEUP_PB0
  - LL_PWR_WAKEUP_PB1
  - LL_PWR_WAKEUP_PB2
  - LL_PWR_WAKEUP_PB3
  - LL_PWR_WAKEUP_PB4
  - LL_PWR_WAKEUP_PB5
  - LL_PWR_WAKEUP_PB6
  - LL_PWR_WAKEUP_PB7
  - LL_PWR_WAKEUP_PB8 (*)
  - LL_PWR_WAKEUP_PB9 (*)
  - LL_PWR_WAKEUP_PB10 (*)
  - LL_PWR_WAKEUP_PB11 (*)
  - LL_PWR_WAKEUP_PB12 (**)
  - LL_PWR_WAKEUP_PB13 (**)
  - LL_PWR_WAKEUP_PB14 (**)
  - LL_PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWU0 LL_PWR_EnableWakeUpPin
- CR3 EWU1 LL_PWR_EnableWakeUpPin
- CR3 EWU2 LL_PWR_EnableWakeUpPin
- CR3 EWU3 LL_PWR_EnableWakeUpPin
- CR3 EWU4 LL_PWR_EnableWakeUpPin
- CR3 EWU5 LL_PWR_EnableWakeUpPin
- CR3 EWU6 LL_PWR_EnableWakeUpPin
- CR3 EWU7 LL_PWR_EnableWakeUpPin
- CR3 EWU8 LL_PWR_EnableWakeUpPin
- CR3 EWU9 LL_PWR_EnableWakeUpPin
- CR3 EWU10 LL_PWR_EnableWakeUpPin
- CR3 EWU11 LL_PWR_EnableWakeUpPin
- CR6 EWU12 LL_PWR_EnableWakeUpPin
- CR6 EWU13 LL_PWR_EnableWakeUpPin
- CR6 EWU14 LL_PWR_EnableWakeUpPin
- CR6 EWU15 LL_PWR_EnableWakeUpPin
- CR6 EWU16 LL_PWR_EnableWakeUpPin
- CR6 EWU17 LL_PWR_EnableWakeUpPin
- CR6 EWU18 LL_PWR_EnableWakeUpPin
- CR6 EWU19 LL_PWR_EnableWakeUpPin
- CR6 EWU20 LL_PWR_EnableWakeUpPin
- CR6 EWU21 LL_PWR_EnableWakeUpPin
- CR6 EWU22 LL_PWR_EnableWakeUpPin
- CR6 EWU23 LL_PWR_EnableWakeUpPin
- CR6 EWU24 LL_PWR_EnableWakeUpPin
- CR6 EWU25 LL_PWR_EnableWakeUpPin
- CR6 EWU26 LL_PWR_EnableWakeUpPin
- CR6 EWU27 LL_PWR_EnableWakeUpPin

LL_PWR_DisableWakeUpPin

Function name

__STATIC_INLINE void LL_PWR_DisableWakeUpPin (uint32_t WakeUpPin)

Function description

Disable Wakeup PINx to get out of DEEPSTOP mode.

Parameters

- **WakeUpPin:** Enable the wakeup sources. This parameter can be a combination of the following values:
  – LL_PWR_WAKEUP_PA0
  – LL_PWR_WAKEUP_PA1
  – LL_PWR_WAKEUP_PA2
  – LL_PWR_WAKEUP_PA3
  – LL_PWR_WAKEUP_PA4 (*)
  – LL_PWR_WAKEUP_PA5 (*)
  – LL_PWR_WAKEUP_PA6 (*)
  – LL_PWR_WAKEUP_PA7 (*)
  – LL_PWR_WAKEUP_PA8
  – LL_PWR_WAKEUP_PA9
  – LL_PWR_WAKEUP_PA10
  – LL_PWR_WAKEUP_PA11
  – LL_PWR_WAKEUP_PA12 (*)
  – LL_PWR_WAKEUP_PA13 (*)
  – LL_PWR_WAKEUP_PA14 (*)
  – LL_PWR_WAKEUP_PA15 (*)
  – LL_PWR_WAKEUP_PB0
  – LL_PWR_WAKEUP_PB1
  – LL_PWR_WAKEUP_PB2
  – LL_PWR_WAKEUP_PB3
  – LL_PWR_WAKEUP_PB4
  – LL_PWR_WAKEUP_PB5
  – LL_PWR_WAKEUP_PB6
  – LL_PWR_WAKEUP_PB7
  – LL_PWR_WAKEUP_PB8 (*)
  – LL_PWR_WAKEUP_PB9 (*)
  – LL_PWR_WAKEUP_PB10 (*)
  – LL_PWR_WAKEUP_PB11 (*)
  – LL_PWR_WAKEUP_PB12 (**)
  – LL_PWR_WAKEUP_PB13 (**)
  – LL_PWR_WAKEUP_PB14 (**)
  – LL_PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWU0 LL_PWR_DisableWakeUpPin
- CR3 EWU1 LL_PWR_DisableWakeUpPin
- CR3 EWU2 LL_PWR_DisableWakeUpPin
- CR3 EWU3 LL_PWR_DisableWakeUpPin
- CR3 EWU4 LL_PWR_DisableWakeUpPin
- CR3 EWU5 LL_PWR_DisableWakeUpPin
- CR3 EWU6 LL_PWR_DisableWakeUpPin
- CR3 EWU7 LL_PWR_DisableWakeUpPin
- CR3 EWU8 LL_PWR_DisableWakeUpPin
- CR3 EWU9 LL_PWR_DisableWakeUpPin
- CR3 EWU10 LL_PWR_DisableWakeUpPin
- CR3 EWU11 LL_PWR_DisableWakeUpPin
- CR6 EWU12 LL_PWR_DisableWakeUpPin
- CR6 EWU13 LL_PWR_DisableWakeUpPin
- CR6 EWU14 LL_PWR_DisableWakeUpPin
- CR6 EWU15 LL_PWR_DisableWakeUpPin
- CR6 EWU16 LL_PWR_DisableWakeUpPin
- CR6 EWU17 LL_PWR_DisableWakeUpPin
- CR6 EWU18 LL_PWR_DisableWakeUpPin
- CR6 EWU19 LL_PWR_DisableWakeUpPin
- CR6 EWU20 LL_PWR_DisableWakeUpPin
- CR6 EWU21 LL_PWR_DisableWakeUpPin
- CR6 EWU22 LL_PWR_DisableWakeUpPin
- CR6 EWU23 LL_PWR_DisableWakeUpPin
- CR6 EWU24 LL_PWR_DisableWakeUpPin
- CR6 EWU25 LL_PWR_DisableWakeUpPin
- CR6 EWU26 LL_PWR_DisableWakeUpPin
- CR6 EWU27 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 a combination of the following values:
  - LL_PWR_WAKEUP_PA0
  - LL_PWR_WAKEUP_PA1
  - LL_PWR_WAKEUP_PA2
  - LL_PWR_WAKEUP_PA3
  - LL_PWR_WAKEUP_PA4 (*)
  - LL_PWR_WAKEUP_PA5 (*)
  - LL_PWR_WAKEUP_PA6 (*)
  - LL_PWR_WAKEUP_PA7 (*)
  - LL_PWR_WAKEUP_PA8
  - LL_PWR_WAKEUP_PA9
  - LL_PWR_WAKEUP_PA10
  - LL_PWR_WAKEUP_PA11
  - LL_PWR_WAKEUP_PA12 (*)
  - LL_PWR_WAKEUP_PA13 (*)
  - LL_PWR_WAKEUP_PA14 (*)
  - LL_PWR_WAKEUP_PA15 (*)
  - LL_PWR_WAKEUP_PB0
  - LL_PWR_WAKEUP_PB1
  - LL_PWR_WAKEUP_PB2
  - LL_PWR_WAKEUP_PB3
  - LL_PWR_WAKEUP_PB4
  - LL_PWR_WAKEUP_PB5
  - LL_PWR_WAKEUP_PB6
  - LL_PWR_WAKEUP_PB7
  - LL_PWR_WAKEUP_PB8 (*)
  - LL_PWR_WAKEUP_PB9 (*)
  - LL_PWR_WAKEUP_PB10 (*)
  - LL_PWR_WAKEUP_PB11 (*)
  - LL_PWR_WAKEUP_PB12 (**)
  - LL_PWR_WAKEUP_PB13 (**)
  - LL_PWR_WAKEUP_PB14 (**)
  - LL_PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

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

Reference Manual to LL API cross reference:

- CR3 EWU0 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU1 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU2 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU3 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU4 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU5 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU6 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU7 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU8 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU9 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU10 LL_PWR_IsEnabledWakeUpPin
- CR3 EWU11 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU12 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU13 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU14 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU15 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU16 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU17 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU18 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU19 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU20 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU21 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU22 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU23 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU24 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU25 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU26 LL_PWR_IsEnabledWakeUpPin
- CR6 EWU27 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:** Pin to configure. This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PA0
  - LL_PWR_WAKEUP_PA1
  - LL_PWR_WAKEUP_PA2
  - LL_PWR_WAKEUP_PA3
  - LL_PWR_WAKEUP_PA4 (*)
  - LL_PWR_WAKEUP_PA5 (*)
  - LL_PWR_WAKEUP_PA6 (*)
  - LL_PWR_WAKEUP_PA7 (*)
  - LL_PWR_WAKEUP_PA8
  - LL_PWR_WAKEUP_PA9
  - LL_PWR_WAKEUP_PA10
  - LL_PWR_WAKEUP_PA11
  - LL_PWR_WAKEUP_PA12 (*)
  - LL_PWR_WAKEUP_PA13 (*)
  - LL_PWR_WAKEUP_PA14 (*)
  - LL_PWR_WAKEUP_PA15 (*)
  - LL_PWR_WAKEUP_PB0
  - LL_PWR_WAKEUP_PB1
  - LL_PWR_WAKEUP_PB2
  - LL_PWR_WAKEUP_PB3
  - LL_PWR_WAKEUP_PB4
  - LL_PWR_WAKEUP_PB5
  - LL_PWR_WAKEUP_PB6
  - LL_PWR_WAKEUP_PB7
  - LL_PWR_WAKEUP_PB8 (*)
  - LL_PWR_WAKEUP_PB9 (*)
  - LL_PWR_WAKEUP_PB10 (*)
  - LL_PWR_WAKEUP_PB11 (*)
  - LL_PWR_WAKEUP_PB12 (**)
  - LL_PWR_WAKEUP_PB13 (**)
  - LL_PWR_WAKEUP_PB14 (**)
  - LL_PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 WUP0 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP1 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP2 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP3 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP4 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP5 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP6 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP7 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP8 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP9 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP10 LL_PWR_SetWakeUpPinPolarityLow
- CR4 WUP11 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP12 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP13 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP14 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP15 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP16 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP17 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP18 LL_PWR_SetWakeUpPinPolarityLow
- CR7 WUP19 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:** Pin to configure. This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PA0
  - LL_PWR_WAKEUP_PA1
  - LL_PWR_WAKEUP_PA2
  - LL_PWR_WAKEUP_PA3
  - LL_PWR_WAKEUP_PA4 (*)
  - LL_PWR_WAKEUP_PA5 (*)
  - LL_PWR_WAKEUP_PA6 (*)
  - LL_PWR_WAKEUP_PA7 (*)
  - LL_PWR_WAKEUP_PA8
  - LL_PWR_WAKEUP_PA9
  - LL_PWR_WAKEUP_PA10
  - LL_PWR_WAKEUP_PA11
  - LL_PWR_WAKEUP_PA12 (*)
  - LL_PWR_WAKEUP_PA13 (*)
  - LL_PWR_WAKEUP_PA14 (*)
  - LL_PWR_WAKEUP_PA15 (*)
  - LL_PWR_WAKEUP_PB0
  - LL_PWR_WAKEUP_PB1
  - LL_PWR_WAKEUP_PB2
  - LL_PWR_WAKEUP_PB3
  - LL_PWR_WAKEUP_PB4
  - LL_PWR_WAKEUP_PB5
  - LL_PWR_WAKEUP_PB6
  - LL_PWR_WAKEUP_PB7
  - LL_PWR_WAKEUP_PB8 (*)
  - LL_PWR_WAKEUP_PB9 (*)
  - LL_PWR_WAKEUP_PB10 (*)
  - LL_PWR_WAKEUP_PB11 (*)
  - LL_PWR_WAKEUP_PB12 (**)
  - LL_PWR_WAKEUP_PB13 (**)
  - LL_PWR_WAKEUP_PB14 (**)
  - LL_PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR4 WUP0 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP1 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP2 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP3 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP4 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP5 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP6 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP7 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP8 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP9 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP10 LL_PWR_SetWakeUpPinPolarityHigh
- CR4 WUP11 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP12 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP13 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP14 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP15 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP16 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP17 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP18 LL_PWR_SetWakeUpPinPolarityHigh
- CR7 WUP19 LL_PWR_SetWakeUpPinPolarityHigh

LL_PWR_IsWakeUpPinPolarityLow

Function name

__STATIC_INLINE uint32_t LL_PWR_IsWakeUpPinPolarityLow (uint32_t WakeUpPin)

Function description

Get the polarity for the I/Os wakeup sources.

Parameters

- **WakeUpPin:** Pin to configure. This parameter can be a combination of the following values:
  - LL_PWR_WAKEUP_PA0
  - LL_PWR_WAKEUP_PA1
  - LL_PWR_WAKEUP_PA2
  - LL_PWR_WAKEUP_PA3
  - LL_PWR_WAKEUP_PA4 (*)
  - LL_PWR_WAKEUP_PA5 (*)
  - LL_PWR_WAKEUP_PA6 (*)
  - LL_PWR_WAKEUP_PA7 (*)
  - LL_PWR_WAKEUP_PA8
  - LL_PWR_WAKEUP_PA9
  - LL_PWR_WAKEUP_PA10
  - LL_PWR_WAKEUP_PA11
  - LL_PWR_WAKEUP_PA12 (*)
  - LL_PWR_WAKEUP_PA13 (*)
  - LL_PWR_WAKEUP_PA14 (*)
  - LL_PWR_WAKEUP_PA15 (*)
  - LL_PWR_WAKEUP_PB0
  - LL_PWR_WAKEUP_PB1
  - LL_PWR_WAKEUP_PB2
  - LL_PWR_WAKEUP_PB3
  - LL_PWR_WAKEUP_PB4
  - LL_PWR_WAKEUP_PB5
  - LL_PWR_WAKEUP_PB6
  - LL_PWR_WAKEUP_PB7
  - LL_PWR_WAKEUP_PB8 (*)
  - LL_PWR_WAKEUP_PB9 (*)
  - LL_PWR_WAKEUP_PB10 (*)
  - LL_PWR_WAKEUP_PB11 (*)
  - LL_PWR_WAKEUP_PB12 (**)
  - LL_PWR_WAKEUP_PB13 (**)
  - LL_PWR_WAKEUP_PB14 (**)
  - LL_PWR_WAKEUP_PB15 (**) (*) available only on STM32WB06 and STM32WB07 devices (**) available only on STM32WB05 and STM32WB09 devices

Return values

- **None:**
- **State:** of bit (1 or 0).

Reference Manual to LL API cross reference:

- CR4 WUP0 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP1 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP2 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP3 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP4 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP5 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP6 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP7 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP8 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP9 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP10 LL_PWR_IsWakeUpPinPolarityLow
- CR4 WUP11 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP12 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP13 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP14 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP15 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP16 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP17 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP18 LL_PWR_IsWakeUpPinPolarityLow
- CR7 WUP19 LL_PWR_IsWakeUpPinPolarityLow

LL_PWR_EnableWU_EWBLE

Function name

__STATIC_INLINE void LL_PWR_EnableWU_EWBLE (void)

Function description

Enable wakeup on BLE event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWBLE LL_PWR_EnableWU_EWBLE

LL_PWR_EnableWU_EWBLEHCPU

Function name

__STATIC_INLINE void LL_PWR_EnableWU_EWBLEHCPU (void)

Function description

Enable wakeup on BLE Host CPU event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWBLEHCPU LL_PWR_EnableWU_EWBLEHCPU

LL_PWR_DisableWU_EWBLE

Function name

__STATIC_INLINE void LL_PWR_DisableWU_EWBLE (void)

Function description

Disable wakeup on BLE Host CPU event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWBLE LL_PWR_DisableWU_EWBLE

LL_PWR_DisableWU_EWBLEHCPU

Function name

__STATIC_INLINE void LL_PWR_DisableWU_EWBLEHCPU (void)

Function description

Disable wakeup on BLE Host CPU event.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR3 EWBLEHCPU LL_PWR_DisableWU_EWBLEHCPU

LL_PWR_IsEnabledWU_EWBLE

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWU_EWBLE (void)

Function description

Check if the wakeup BLE is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CR3 EWBLE LL_PWR_IsEnabledWU_EWBLE

LL_PWR_IsEnabledWU_EWBLEHCPU

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledWU_EWBLEHCPU (void)

Function description

Check if the wakeup BLE Host CPU is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CR3 EWBLEHCPU LL_PWR_IsEnabledWU_EWBLEHCPU

LL_PWR_IsActiveFlag_InternWU

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_InternWU (void)

Function description

Get Internal Wake-up line Flag (RTC)

Return values

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

Reference Manual to LL API cross reference:

- SR1 IWUF LL_PWR_IsActiveFlag_InternWU

LL_PWR_IsActiveFlag_InternWU2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_InternWU2 (void)

Function description

Get Internal Wake-up line Flag (LPUART)

Return values

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

Reference Manual to LL API cross reference:

- SR1 IWUF2 LL_PWR_IsActiveFlag_InternWU2

LL_PWR_IsActiveFlag_WU0

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU0 (void)

Function description

Get Wake-up Flag 0.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WUF0 LL_PWR_IsActiveFlag_WU0

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_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_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_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_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_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_WU7

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU7 (void)

Function description

Get Wake-up Flag 7.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WUF7 LL_PWR_IsActiveFlag_WU7

LL_PWR_IsActiveFlag_WU8

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU8 (void)

Function description

Get Wake-up Flag 8.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WUF8 LL_PWR_IsActiveFlag_WU8

LL_PWR_IsActiveFlag_WU9

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU9 (void)

Function description

Get Wake-up Flag 9.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WUF9 LL_PWR_IsActiveFlag_WU9

LL_PWR_IsActiveFlag_WU10

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU10 (void)

Function description

Get Wake-up Flag 10.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WUF10 LL_PWR_IsActiveFlag_WU10

LL_PWR_IsActiveFlag_WU11

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU11 (void)

Function description

Get Wake-up Flag 11.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WUF11 LL_PWR_IsActiveFlag_WU11

LL_PWR_IsActiveFlag_WU12

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU12 (void)

Function description

Get Wake-up Flag 12.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF12 LL_PWR_IsActiveFlag_WU12

LL_PWR_IsActiveFlag_WU13

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU13 (void)

Function description

Get Wake-up Flag 13.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF13 LL_PWR_IsActiveFlag_WU13

LL_PWR_IsActiveFlag_WU14

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU14 (void)

Function description

Get Wake-up Flag 14.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF14 LL_PWR_IsActiveFlag_WU14

LL_PWR_IsActiveFlag_WU15

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU15 (void)

Function description

Get Wake-up Flag 15.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF15 LL_PWR_IsActiveFlag_WU15

LL_PWR_IsActiveFlag_WU16

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU16 (void)

Function description

Get Wake-up Flag 16.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF16 LL_PWR_IsActiveFlag_WU16

LL_PWR_IsActiveFlag_WU17

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU17 (void)

Function description

Get Wake-up Flag 17.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF17 LL_PWR_IsActiveFlag_WU17

LL_PWR_IsActiveFlag_WU18

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU18 (void)

Function description

Get Wake-up Flag 18.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF18 LL_PWR_IsActiveFlag_WU18

LL_PWR_IsActiveFlag_WU19

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WU19 (void)

Function description

Get Wake-up Flag 19.

Return values

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

Reference Manual to LL API cross reference:

- SR3 WUF19 LL_PWR_IsActiveFlag_WU19

LL_PWR_GetWakeupSource

Function name

__STATIC_INLINE uint32_t LL_PWR_GetWakeupSource (void)

Function description

Get which source woken up the device after a DEEPSTOP.

Return values

- **Wakeup:** Source from DEEPSTOP. This parameter can be a combination of the following values:
  - LL_PWR_SR1_WUFI
  - LL_PWR_SR1_WUFI2 (**)
  - LL_PWR_SR1_BHWF
  - LL_PWR_SR1_WBLEF
  - LL_PWR_SR1_WUF0
  - LL_PWR_SR1_WUF1
  - LL_PWR_SR1_WUF2
  - LL_PWR_SR1_WUF3
  - LL_PWR_SR1_WUF4
  - LL_PWR_SR1_WUF5
  - LL_PWR_SR1_WUF6
  - LL_PWR_SR1_WUF7
  - LL_PWR_SR1_WUF8
  - LL_PWR_SR1_WUF9
  - LL_PWR_SR1_WUF10
  - LL_PWR_SR1_WUF11
  - LL_PWR_SR3_WUF12
  - LL_PWR_SR3_WUF13
  - LL_PWR_SR3_WUF14
  - LL_PWR_SR3_WUF15
  - LL_PWR_SR3_WUF16
  - LL_PWR_SR3_WUF17
  - LL_PWR_SR3_WUF18
  - LL_PWR_SR3_WUF19
  - LL_PWR_SR3_WUF20 (*)
  - LL_PWR_SR3_WUF21 (*)
  - LL_PWR_SR3_WUF22 (*)
  - LL_PWR_SR3_WUF23 (*)
  - LL_PWR_SR3_WUF24 (*)
  - LL_PWR_SR3_WUF25 (*)
  - LL_PWR_SR3_WUF26 (*)
  - LL_PWR_SR3_WUF27 (*) (*) Not available on devices STM32WB05 and STM32WB09 (**) Not available on devices STM32WB07

Reference Manual to LL API cross reference:

- SR1 WUFI LL_PWR_GetWakeupSource
- SR1 WUFI2 LL_PWR_GetWakeupSource
- SR1 BHWF LL_PWR_GetWakeupSource
- SR1 WBLEF LL_PWR_GetWakeupSource
- SR1 WUF0 LL_PWR_GetWakeupSource
- SR1 WUF1 LL_PWR_GetWakeupSource
- SR1 WUF2 LL_PWR_GetWakeupSource
- SR1 WUF3 LL_PWR_GetWakeupSource
- SR1 WUF4 LL_PWR_GetWakeupSource
- SR1 WUF5 LL_PWR_GetWakeupSource
- SR1 WUF6 LL_PWR_GetWakeupSource
- SR1 WUF7 LL_PWR_GetWakeupSource
- SR1 WUF8 LL_PWR_GetWakeupSource
- SR1 WUF9 LL_PWR_GetWakeupSource
- SR1 WUF10 LL_PWR_GetWakeupSource
- SR1 WUF11 LL_PWR_GetWakeupSource
- SR3 WUF12 LL_PWR_GetWakeupSource
- SR3 WUF13 LL_PWR_GetWakeupSource
- SR3 WUF14 LL_PWR_GetWakeupSource
- SR3 WUF15 LL_PWR_GetWakeupSource
- SR3 WUF16 LL_PWR_GetWakeupSource
- SR3 WUF17 LL_PWR_GetWakeupSource
- SR3 WUF18 LL_PWR_GetWakeupSource
- SR3 WUF19 LL_PWR_GetWakeupSource
- SR3 WUF20 LL_PWR_GetWakeupSource
- SR3 WUF21 LL_PWR_GetWakeupSource
- SR3 WUF22 LL_PWR_GetWakeupSource
- SR3 WUF23 LL_PWR_GetWakeupSource
- SR3 WUF24 LL_PWR_GetWakeupSource
- SR3 WUF25 LL_PWR_GetWakeupSource
- SR3 WUF26 LL_PWR_GetWakeupSource
- SR3 WUF27 LL_PWR_GetWakeupSource

LL_PWR_ClearWakeupSource

Function name

__STATIC_INLINE void LL_PWR_ClearWakeupSource (uint32_t source)

Function description

Clear Wake-up Flags.

Parameters

- **source:** Wakeup Source from DEEPSTOP. This parameter can be a combination of the following values:
  - LL_PWR_SR1_WUFI2 (**)
  - LL_PWR_SR1_BHWF
  - LL_PWR_SR1_WBLEF
  - LL_PWR_SR1_WUF0
  - LL_PWR_SR1_WUF1
  - LL_PWR_SR1_WUF2
  - LL_PWR_SR1_WUF3
  - LL_PWR_SR1_WUF4
  - LL_PWR_SR1_WUF5
  - LL_PWR_SR1_WUF6
  - LL_PWR_SR1_WUF7
  - LL_PWR_SR1_WUF8
  - LL_PWR_SR1_WUF9
  - LL_PWR_SR1_WUF10
  - LL_PWR_SR1_WUF11
  - LL_PWR_SR3_WUF12
  - LL_PWR_SR3_WUF13
  - LL_PWR_SR3_WUF14
  - LL_PWR_SR3_WUF15
  - LL_PWR_SR3_WUF16
  - LL_PWR_SR3_WUF17
  - LL_PWR_SR3_WUF18
  - LL_PWR_SR3_WUF19
  - LL_PWR_SR3_WUF20 (*)
  - LL_PWR_SR3_WUF21 (*)
  - LL_PWR_SR3_WUF22 (*)
  - LL_PWR_SR3_WUF23 (*)
  - LL_PWR_SR3_WUF24 (*)
  - LL_PWR_SR3_WUF25 (*)
  - LL_PWR_SR3_WUF26 (*)
  - LL_PWR_SR3_WUF27 (*) (*) Not available on devices STM32WB05 and STM32WB09 (**) Not available on devices STM32WB07

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUFI2 LL_PWR_ClearFlag_WU
- SR1 BHWF LL_PWR_ClearFlag_WU
- SR1 WBLEF LL_PWR_ClearFlag_WU
- SR1 WUF0 LL_PWR_ClearFlag_WU
- SR1 WUF1 LL_PWR_ClearFlag_WU
- SR1 WUF2 LL_PWR_ClearFlag_WU
- SR1 WUF3 LL_PWR_ClearFlag_WU
- SR1 WUF4 LL_PWR_ClearFlag_WU
- SR1 WUF5 LL_PWR_ClearFlag_WU
- SR1 WUF6 LL_PWR_ClearFlag_WU
- SR1 WUF7 LL_PWR_ClearFlag_WU
- SR1 WUF8 LL_PWR_ClearFlag_WU
- SR1 WUF9 LL_PWR_ClearFlag_WU
- SR1 WUF10 LL_PWR_ClearFlag_WU
- SR1 WUF11 LL_PWR_ClearFlag_WU
- SR3 WUF12 LL_PWR_ClearFlag_WU
- SR3 WUF13 LL_PWR_ClearFlag_WU
- SR3 WUF14 LL_PWR_ClearFlag_WU
- SR3 WUF15 LL_PWR_ClearFlag_WU
- SR3 WUF16 LL_PWR_ClearFlag_WU
- SR3 WUF17 LL_PWR_ClearFlag_WU
- SR3 WUF18 LL_PWR_ClearFlag_WU
- SR3 WUF19 LL_PWR_ClearFlag_WU
- SR3 WUF20 LL_PWR_ClearFlag_WU
- SR3 WUF21 LL_PWR_ClearFlag_WU
- SR3 WUF22 LL_PWR_ClearFlag_WU
- SR3 WUF23 LL_PWR_ClearFlag_WU
- SR3 WUF24 LL_PWR_ClearFlag_WU
- SR3 WUF25 LL_PWR_ClearFlag_WU
- SR3 WUF26 LL_PWR_ClearFlag_WU
- SR3 WUF27 LL_PWR_ClearFlag_WU

LL_PWR_ClearFlag_WU0

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU0 ()

Function description

Clear Wake-up Flag 0.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF0 LL_PWR_ClearFlag_WU0

LL_PWR_ClearFlag_WU1

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU1 ()

Function description

Clear Wake-up Flag 1.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF1 LL_PWR_ClearFlag_WU1

LL_PWR_ClearFlag_WU2

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU2 ()

Function description

Clear Wake-up Flag 2.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF2 LL_PWR_ClearFlag_WU2

LL_PWR_ClearFlag_WU3

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU3 ()

Function description

Clear Wake-up Flag 3.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF3 LL_PWR_ClearFlag_WU3

LL_PWR_ClearFlag_WU4

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU4 ()

Function description

Clear Wake-up Flag 4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF4 LL_PWR_ClearFlag_WU4

LL_PWR_ClearFlag_WU5

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU5 ()

Function description

Clear Wake-up Flag 5.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF5 LL_PWR_ClearFlag_WU5

LL_PWR_ClearFlag_WU6

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU6 ()

Function description

Clear Wake-up Flag 6.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF6 LL_PWR_ClearFlag_WU6

LL_PWR_ClearFlag_WU7

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU7 ()

Function description

Clear Wake-up Flag 7.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF5 LL_PWR_ClearFlag_WU7

LL_PWR_ClearFlag_WU8

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU8 ()

Function description

Clear Wake-up Flag 8.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF5 LL_PWR_ClearFlag_WU8

LL_PWR_ClearFlag_WU9

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU9 ()

Function description

Clear Wake-up Flag 9.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF9 LL_PWR_ClearFlag_WU9

LL_PWR_ClearFlag_WU10

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU10 ()

Function description

Clear Wake-up Flag 10.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF10 LL_PWR_ClearFlag_WU10

LL_PWR_ClearFlag_WU11

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU11 ()

Function description

Clear Wake-up Flag 11.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WUF11 LL_PWR_ClearFlag_WU11

LL_PWR_ClearFlag_WU12

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU12 ()

Function description

Clear Wake-up Flag 12.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF12 LL_PWR_ClearFlag_WU12

LL_PWR_ClearFlag_WU13

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU13 ()

Function description

Clear Wake-up Flag 13.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF13 LL_PWR_ClearFlag_WU13

LL_PWR_ClearFlag_WU14

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU14 ()

Function description

Clear Wake-up Flag 14.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF14 LL_PWR_ClearFlag_WU14

LL_PWR_ClearFlag_WU15

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU15 ()

Function description

Clear Wake-up Flag 15.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF15 LL_PWR_ClearFlag_WU15

LL_PWR_ClearFlag_WU16

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU16 ()

Function description

Clear Wake-up Flag 16.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF16 LL_PWR_ClearFlag_WU16

LL_PWR_ClearFlag_WU17

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU17 ()

Function description

Clear Wake-up Flag 17.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF17 LL_PWR_ClearFlag_WU17

LL_PWR_ClearFlag_WU18

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU18 ()

Function description

Clear Wake-up Flag 18.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF18 LL_PWR_ClearFlag_WU18

LL_PWR_ClearFlag_WU19

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WU19 ()

Function description

Clear Wake-up Flag 19.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR3 WUF19 LL_PWR_ClearFlag_WU19

LL_PWR_IsActiveFlag_WBLEHCPUF

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WBLEHCPUF (void)

Function description

Get BLE wakeup interrupt flag.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WBLEHCPUF LL_PWR_IsActiveFlag_WBLEHCPUF

LL_PWR_IsActiveFlag_WBLEF

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_WBLEF (void)

Function description

Get BLE wakeup interrupt flag.

Return values

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

Reference Manual to LL API cross reference:

- SR1 WBLEF LL_PWR_IsActiveFlag_WBLEF

LL_PWR_ClearFlag_WBLEHCPUF

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WBLEHCPUF (void)

Function description

Clear BLE HOST CPU wakeup interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WBLEHCPUF LL_PWR_ClearFlag_WBLEHCPU

LL_PWR_ClearFlag_WBLEF

Function name

__STATIC_INLINE void LL_PWR_ClearFlag_WBLEF (void)

Function description

Clear BLE wakeup interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SR1 WBLEF LL_PWR_ClearFlag_WBLEF

LL_PWR_GetRFWakeupFlag

Function name

__STATIC_INLINE uint32_t LL_PWR_GetRFWakeupFlag (void)

Function description

Check if a radio wakeup event occurs.

Return values

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

Reference Manual to LL API cross reference:

- EXTSRR RFPHASEF LL_PWR_GetRFWakeupFlag

LL_PWR_ClearRFWakeupFlag

Function name

__STATIC_INLINE void LL_PWR_ClearRFWakeupFlag (void)

Function description

Clear radio wakeup event occurs flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EXTSRR RFPHASEF LL_PWR_ClearRFWakeupFlag

LL_PWR_GetDeepstopSeqFlag

Function name

__STATIC_INLINE uint32_t LL_PWR_GetDeepstopSeqFlag (void)

Function description

Check if a DEEPSTOP sequence occurred.

Return values

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

Reference Manual to LL API cross reference:

- EXTSRR DEEPSTOPF LL_PWR_GetDeepstopSeqFlag

LL_PWR_ClearDeepstopSeqFlag

Function name

__STATIC_INLINE void LL_PWR_ClearDeepstopSeqFlag (void)

Function description

Clear DEEPSTOP sequence occurred flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- EXTSRR DEEPSTOPF LL_PWR_ClearRFWakeupFlag

LL_PWR_GetIOBootVal

Function name

__STATIC_INLINE uint32_t LL_PWR_GetIOBootVal (uint32_t IO)

Function description

Get IO BOOT value.

Parameters

- **IO:** This parameter can be one of the following values:
  – LL_PWR_PA8_LATCH_POR
  – LL_PWR_PA9_LATCH_POR
  – LL_PWR_PA10_LATCH_POR
  – LL_PWR_PA11_LATCH_POR
  – LL_PWR_PB12_LATCH_POR (*)
  – LL_PWR_PB13_LATCH_POR (*)
  – LL_PWR_PB14_LATCH_POR (*)
  – LL_PWR_PB15_LATCH_POR (*) (*) Not available on device STM32WB05

Return values

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

Reference Manual to LL API cross reference:

- SR2 LL_PWR_GetIOBootVal

LL_PWR_IsActiveFlag_PVDO

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_PVDO (void)

Function description

Indicate whether VDD voltage is below or above the selected PVD threshold.

Return values

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

Reference Manual to LL API cross reference:

- SR2 PVDO LL_PWR_IsActiveFlag_PVDO

LL_PWR_IsActiveFlag_REGLPS

Function name

__STATIC_INLINE uint32_t LL_PWR_IsActiveFlag_REGLPS (void)

Function description

Indicate whether or not the low-power regulator is ready.

Return values

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

Reference Manual to LL API cross reference:

- SR2 REGLPS LL_PWR_IsActiveFlag_REGLPS

LL_PWR_IsSMPSReady

Function name

__STATIC_INLINE uint32_t LL_PWR_IsSMPSReady (void)

Function description

Get the SMPS ready status.

Return values

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

Reference Manual to LL API cross reference:

- SR2 SMPSRDY LL_PWR_IsSMPSReady

LL_PWR_IsSMPSinRUNMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsSMPSinRUNMode (void)

Function description

Get the SMPS RUN mode status.

Return values

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

Reference Manual to LL API cross reference:

- SR2 SMPSENR LL_PWR_IsSMPSinRUNMode

LL_PWR_IsSMPSinPRECHARGEMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsSMPSinPRECHARGEMode (void)

Function description

Get the SMPS PRECHARGE mode status.

Return values

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

Reference Manual to LL API cross reference:

- SR2 SMPSBYPR LL_PWR_IsSMPSinPRECHARGEMode

LL_PWR_SetSMPSMode

Function name

__STATIC_INLINE void LL_PWR_SetSMPSMode (uint32_t mode)

Function description

Set SMPS Mode.

Parameters

- **mode:** SMPS Mode. This parameter can be one of the following values:
  – LL_PWR_SMPS
  – LL_PWR_NO_SMPS

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 NOSMPS LL_PWR_SetSMPSMode

LL_PWR_GetSMPSMode

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSMPSMode (void)

Function description

Get SMPS Mode.

Return values

- **SMPS:** Mode. This parameter can be one of the following values:
  – LL_PWR_SMPS
  – LL_PWR_NO_SMPS

Reference Manual to LL API cross reference:

- CR5 NOSMPS LL_PWR_GetSMPSMode

LL_PWR_SetSMPSPrechargeMode

Function name

__STATIC_INLINE void LL_PWR_SetSMPSPrechargeMode (uint32_t mode)

Function description

Set SMPS in PRECHARGE Mode.

Parameters

- **mode:** SMPS in Precharge Mode. This parameter can be one of the following values:
  – LL_PWR_NO_SMPS_PRECHARGE
  – LL_PWR_SMPS_PRECHARGE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 SMPSFBYP LL_PWR_SetSMPSPrechargeMode

LL_PWR_IsEnabledSMPSPrechargeMode

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledSMPSPrechargeMode (void)

Function description

Check if SMPS is configured in PRECHARGE Mode.

Return values

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

Reference Manual to LL API cross reference:

- CR5 SMPSFBYP LL_PWR_IsEnabledSMPSPrechargeMode

LL_PWR_SetSMPSOpenMode

Function name

__STATIC_INLINE void LL_PWR_SetSMPSOpenMode (uint32_t mode)

Function description

Select OPEN Mode for SMPS during DEEPSTOP.

Parameters

- **mode:** SMPS Mode. This parameter can be one of the following values:
  – LL_PWR_NO_SMPS_LPOPEN
  – LL_PWR_SMPS_LPOPEN

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 SMPSLPOPEN LL_PWR_SetSMPSOpenMode

LL_PWR_SetSMPSBOM

Function name

__STATIC_INLINE void LL_PWR_SetSMPSBOM (uint32_t BOM)

Function description

Set SMPS BOM.

Parameters

- **BOM:** BOM. This parameter can be one of the following values:
  - LL_PWR_SMPS_BOM1
  - LL_PWR_SMPS_BOM2
  - LL_PWR_SMPS_BOM3

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 SMPSBOMSEL LL_PWR_SetSMPSBOM CR5 SMPSBOMSEL LL_PWR_SetSMPSBOM

LL_PWR_GetSMPSBOM

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSMPSBOM (void)

Function description

Get SMPS BOM.

Return values

- **BOM.:** This parameter can be one of the following values:
  - LL_PWR_SMPS_BOM1
  - LL_PWR_SMPS_BOM2
  - LL_PWR_SMPS_BOM3

Reference Manual to LL API cross reference:

- CR5 SMPSBOMSEL LL_PWR_GetSMPSBOM CR5 SMPSBOMSEL LL_PWR_GetSMPSBOM

LL_PWR_SMPS_SetOutputVoltageLevel

Function name

__STATIC_INLINE void LL_PWR_SMPS_SetOutputVoltageLevel (uint32_t OutputVoltageLevel)

Function description

Set SMPS Output Level.

Parameters

- **OutputVoltageLevel:** Output Level. This parameter can be one of the following values:
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V20
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V25
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V30
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V35
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V40
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V45
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V50
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V55
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V60
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V65
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V70
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V75
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V80
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V85
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V90
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V95

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 SMPSLVL LL_PWR_SMPS_SetOutputVoltageLevel

LL_PWR_SMPS_GetOutputVoltageLevel

Function name

__STATIC_INLINE uint32_t LL_PWR_SMPS_GetOutputVoltageLevel (void)

Function description

Get SMPS Output Level.

Return values

- **Output:** Level. This parameter can be one of the following values:
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V20
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V25
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V30
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V35
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V40
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V45
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V50
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V55
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V60
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V65
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V70
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V75
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V80
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V85
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V90
  - LL_PWR_SMPS_OUTPUT_VOLTAGE_1V95

Reference Manual to LL API cross reference:

- CR5 SMPSLVL LL_PWR_SMPS_GetOutputVoltageLevel

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 Shutdown mode.

Parameters

- **GPIO:** This parameter can be one of the following values:
  – LL_PWR_GPIO_A
  – LL_PWR_GPIO_B
- **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

- Some pins are not configurable for pulling in Shutdown modes. Refer to reference manual for available pins.

Reference Manual to LL API cross reference:

- PUCRA PUA0-15 LL_PWR_EnableGPIOPullUp
- PUCRB PUB0-15 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 Shutdown mode.

Parameters

- **GPIO:** This parameter can be one of the following values:
  – LL_PWR_GPIO_A
  – LL_PWR_GPIO_B
- **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

- Some pins are not configurable for pulling in Shutdown modes. Refer to reference manual for available pins.

Reference Manual to LL API cross reference:

- PUCRA PUA0-15 LL_PWR_DisableGPIOPullUp
- PUCRB PUB0-15 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
- **GPIONumber:** This parameter can be a combination 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)

Reference Manual to LL API cross reference:

- PUCRA PUA0-15 LL_PWR_IsEnabledGPIOPullUp
- PUCRB PUB0-15 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 Shutdown mode.

Parameters

- **GPIO:** This parameter can be one of the following values:
  – LL_PWR_GPIO_A
  – LL_PWR_GPIO_B
- **GPIONumber:** This parameter can be a combination 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

- Some pins are not configurable for pulling in Shutdown modes. Refer to reference manual for available pins

Reference Manual to LL API cross reference:

- PDCRA PDA0-15 LL_PWR_EnableGPIOPullDown
- PDCRB PUB0-15 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 Shutdown mode.

Parameters

- **GPIO:** This parameter can be one of the following values:
  - LL_PWR_GPIO_A
  - LL_PWR_GPIO_B
- **GPIONumber:** This parameter can be a combination 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

- Some pins are not configurable for pulling in Shutdown modes. Refer to reference manual for available pins.

Reference Manual to LL API cross reference:

- PDCRA PDA0-15 LL_PWR_DisableGPIOPullDown
- PDCRB PDB0-15 LL_PWR_DisableGPIOPullDown

LL_PWR_IsEnabledGPIOPullDown

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledGPIOPullDown (uint32_t GPIO, uint32_t GPIONumber)

Function description

Checks 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
- **GPIONumber:** This parameter can be a combination 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)

Reference Manual to LL API cross reference:

- PDCRA PDA0-15 LL_PWR_IsEnabledGPIOPullDown
- PDCRB PDB0-15 LL_PWR_IsEnabledGPIOPullDown

LL_PWR_SetNoPullA

Function name

__STATIC_INLINE void LL_PWR_SetNoPullA (uint32_t GPIONumber)

Function description

Configure the GPIO in NO PULL configuration for port A.

Parameters

- **GPIONumber:** Pins No Pull configuration. This parameter can be a combination 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(*) (*) available only on STM32WB06 and STM32WB07 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- PDCRA/PUCRA LL_PWR_SetNoPullA

LL_PWR_SetNoPullB

Function name

__STATIC_INLINE void LL_PWR_SetNoPullB (uint32_t GPIONumber)

Function description

Configure the GPIO in NO PULL configuration for port B.

Parameters

- **GPIONumber:** Pins No Pull configuration. This parameter can be a combination 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 (*) available only oon STM32WB06 and STM32WB07 devices

Return values

- **None:**

Reference Manual to LL API cross reference:

- PDCRB/PUCRB LL_PWR_SetNoPullB

LL_PWR_SetSMPSTrim

Function name

__STATIC_INLINE void LL_PWR_SetSMPSTrim (uint32_t trim)

Function description

Set SMPS trimming value.

Parameters

- **trim:** SMPS output voltage trimming value

Return values

- **None:**

Reference Manual to LL API cross reference:

- ENGTRIM LL_PWR_SetSMPSTrim

LL_PWR_GetSMPSTrim

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSMPSTrim (void)

Function description

Get SMPS trimming value.

Return values

- **SMPS:** output voltage trimming value

Reference Manual to LL API cross reference:

- TRIMR/ENGTRIM LL_PWR_GetSMPSTrim

LL_PWR_SetMRTrim

Function name

__STATIC_INLINE void LL_PWR_SetMRTrim (uint32_t trim)

Function description

Set Main regulator voltage trimming value.

Parameters

- **trim:** Main Regulator voltage trimming value

Return values

- **None:**

Reference Manual to LL API cross reference:

- ENGTRIM LL_PWR_SetMRTrim

LL_PWR_GetMRTrim

Function name

__STATIC_INLINE uint32_t LL_PWR_GetMRTrim (void)

Function description

Get Main regulator voltage trimming value.

Return values

- **Main:** regulator voltage trimming value

Reference Manual to LL API cross reference:

- TRIMR/ENGTRIM LL_PWR_GetMRTrim

LL_PWR_EnableDEEPSTOP2

Function name

__STATIC_INLINE void LL_PWR_EnableDEEPSTOP2 (void)

Function description

Enable the DEEPSTOP2 feature, debugger feature not lost during DEEPSTOP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGR DEEPSTOP2 LL_PWR_EnableDEEPSTOP2

LL_PWR_DisableDEEPSTOP2

Function name

__STATIC_INLINE void LL_PWR_DisableDEEPSTOP2 (void)

Function description

Disable the DEEPSTOP2 feature, debugger feature lost during DEEPSTOP.

Return values

- **None:**

Reference Manual to LL API cross reference:

- DBGR DEEPSTOP2 LL_PWR_DisableDEEPSTOP2

LL_PWR_IsEnabledDEEPSTOP2

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledDEEPSTOP2 (void)

Function description

Checks if the DEEPSTOP2 feature is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- DBGR DEEPSTOP2 LL_PWR_IsEnabledDEEPSTOP2

LL_PWR_SetSMPSPrechargeLimitCurrent

Function name

__STATIC_INLINE void LL_PWR_SetSMPSPrechargeLimitCurrent (uint32_t mode)

Function description

Select SMPS PRECHARGE limit current.

Parameters

- **mode:** This parameter can be one of the following values:
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_2_5
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_5
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_10
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_20

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR5 SMPS_PRECH_CUR_SEL LL_PWR_SetSMPSPrechargeLimitCurrent

LL_PWR_GetSMPSPrechargeLimitCurrent

Function name

__STATIC_INLINE uint32_t LL_PWR_GetSMPSPrechargeLimitCurrent (void)

Function description

Get SMPS PRECHARGE limit current selected.

Return values

- **This:** parameter can be one of the following values:
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_2_5
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_5
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_10
  – LL_PWR_SMPS_PRECH_LIMIT_CUR_20

Reference Manual to LL API cross reference:

- CR5 SMPS_PRECH_CUR_SEL LL_PWR_GetSMPSPrechargeLimitCurrent

LL_PWR_EnableIOWakeupSDN

Function name

__STATIC_INLINE void LL_PWR_EnableIOWakeupSDN (void)

Function description

Shutdown I/O Wakeup enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SDWN_WUEN WUEN LL_PWR_EnableIOWakeupSDN

LL_PWR_DisableIOWakeupSDN

Function name

__STATIC_INLINE void LL_PWR_DisableIOWakeupSDN (void)

Function description

Shutdown I/O Wakeup disable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SDWN_WUEN WUEN LL_PWR_DisableIOWakeupSDN

LL_PWR_IsEnabledIOWakeupSDN

Function name

__STATIC_INLINE uint32_t LL_PWR_IsEnabledIOWakeupSDN (void)

Function description

Checks if Shutdown I/O Wakeup is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- SDWN_WUEN WUEN LL_PWR_IsEnabledIOWakeupSDN

LL_PWR_IOWakeupPolaritySDN

Function name

__STATIC_INLINE void LL_PWR_IOWakeupPolaritySDN (uint8_t pol)

Function description

Shutdown I/O Wakeup Polarity configuration.

Parameters

- **pol:**
  - 0 : PB0 pin wakeup from shutdown on high pulse or high level detection
  - 1 : PB0 pin wakeup from shutdown on low pulse or low level detection

Return values

- **None:**

Reference Manual to LL API cross reference:

- SDWN_WUPOL WUPOL LL_PWR_IOWakeupPolaritySDN

LL_PWR_IsIOWakeupSDN

Function name

__STATIC_INLINE uint32_t LL_PWR_IsIOWakeupSDN (void)

Function description

Checks if Shutdown I/O Wakeup from PB0 pin occurred.

Return values

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

Reference Manual to LL API cross reference:

- SDWN_WUF WUF LL_PWR_IsIOWakeupSDN

LL_PWR_ClearIOWakeupFlagSDN

Function name

__STATIC_INLINE void LL_PWR_ClearIOWakeupFlagSDN (void)

Function description

Clear I/O Wakeup from PB0 pin occurred flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- SDWN_WUF WUF LL_PWR_ClearIOWakeupFlagSDN

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

56.2 PWR Firmware driver defines

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

56.2.1 PWR

PWR

Get Flags Defines

LL_PWR_SR1_WUFI

LL_PWR_SR1_WUFI2

LL_PWR_SR1_BHWF

LL_PWR_SR1_WBLEF

LL_PWR_SR1_WUF0

LL_PWR_SR1_WUF1

LL_PWR_SR1_WUF2

LL_PWR_SR1_WUF3

LL_PWR_SR1_WUF4

LL_PWR_SR1_WUF5

LL_PWR_SR1_WUF6

LL_PWR_SR1_WUF7

LL_PWR_SR1_WUF8

LL_PWR_SR1_WUF9

LL_PWR_SR1_WUF10

LL_PWR_SR1_WUF11

LL_PWR_SR3_WUF12

LL_PWR_SR3_WUF13

LL_PWR_SR3_WUF14

LL_PWR_SR3_WUF15

LL_PWR_SR3_WUF16

LL_PWR_SR3_WUF17

LL_PWR_SR3_WUF18

LL_PWR_SR3_WUF19

LL_PWR_SR1_WUFALL

LL_PWR_SR3_WUFALL

LL_PWR_WUFALL

LL_PWR_EXTSRR_RFPHASEF

LL_PWR_EXTSRR_DEEPSTOPF

GPIO Port

LL_PWR_GPIO_A

LL_PWR_GPIO_B

PWR GPIO Pin Mask

LL_PWR_GPIO_BIT_0

GPIO port I/O pin 0

LL_PWR_GPIO_BIT_1

GPIO port I/O pin 1

LL_PWR_GPIO_BIT_2

GPIO port I/O pin 2

LL_PWR_GPIO_BIT_3

GPIO port I/O pin 3

LL_PWR_GPIO_BIT_4

GPIO port I/O pin 4

LL_PWR_GPIO_BIT_5

GPIO port I/O pin 5

LL_PWR_GPIO_BIT_6

GPIO port I/O pin 6

LL_PWR_GPIO_BIT_7
GPIO port I/O pin 7

LL_PWR_GPIO_BIT_8
GPIO port I/O pin 8

LL_PWR_GPIO_BIT_9
GPIO port I/O pin 9

LL_PWR_GPIO_BIT_10
GPIO port I/O pin 10

LL_PWR_GPIO_BIT_11
GPIO port I/O pin 11

LL_PWR_GPIO_BIT_12
GPIO port I/O pin 12

LL_PWR_GPIO_BIT_13
GPIO port I/O pin 13

LL_PWR_GPIO_BIT_14
GPIO port I/O pin 14

LL_PWR_GPIO_BIT_15
GPIO port I/O pin 15

IOs PU/PD configuration

LL_PWR_WUP_RISIEDG
Detection of wakeup event on rising edge.

LL_PWR_WUP_FALLEDG
Detection of wakeup event on falling edge.

PA8-PA11 I/Os value latched at POR.

LL_PWR_PA8_LATCH_POR
PA8 input value latched at POR.

LL_PWR_PA9_LATCH_POR
PA9 input value latched at POR.

LL_PWR_PA10_LATCH_POR
PA10 input value latched at POR.

LL_PWR_PA11_LATCH_POR
PA11 input value latched at POR.

PB12-PB15 I/Os value latched at POR.

LL_PWR_PB12_LATCH_POR
PB12 input value latched at POR.

LL_PWR_PB13_LATCH_POR
PB13 input value latched at POR.

LL_PWR_PB14_LATCH_POR

PB14 input value latched at POR.

LL_PWR_PB15_LATCH_POR

PB15 input value latched at POR.

PA4-PA11 output configuration during DEEPSTOP

LL_PWR_IOCFG_BYP

BYPASS mode. The I/O mode is controlled by the GPIO block registers in active mode and switch to input mode during DEESPTOP state.

LL_PWR_IOCFG_RTC

RTC_OUT signal for PA11 and PA8

LL_PWR_IOCFG_LCO

LCO output signal for PA10 and PA9

LL_PWR_IOCFG_LOW

I/O drives a low level in both active and DEEPSTOP modes

LL_PWR_IOCFG_HIGH

I/O drives a high level in both active and DEEPSTOP modes

PWR Mode

LL_PWR_MODE_DEEPSTOP

LL_PWR_MODE_SHUTDOWN

PVD Level

LL_PWR_PVDLEVEL_0

VPVD0 around 2.0 V

LL_PWR_PVDLEVEL_1

VPVD1 around 2.2 V

LL_PWR_PVDLEVEL_2

VPVD2 around 2.4 V

LL_PWR_PVDLEVEL_3

VPVD3 around 2.5 V

LL_PWR_PVDLEVEL_4

VPVD4 around 2.6 V

LL_PWR_PVDLEVEL_5

VPVD5 around 2.8 V

LL_PWR_PVDLEVEL_6

VPVD6 around 2.9 V

LL_PWR_PVDLEVEL_7

External input analog voltage (Compare internally to VBGP)

RAM Retention enable

LL_PWR_RAMRET_1

LL_PWR_RAMRET_2

LL_PWR_RAMRET_3

SMPS BOM

LL_PWR_SMPS_BOM1

LL_PWR_SMPS_BOM2

LL_PWR_SMPS_BOM3

SMPS Mode

LL_PWR_SMPS

SMPS is enabled

LL_PWR_NO_SMPS

SMPS is disabled

SMPS OPEN Mode during DEEPSTOP

LL_PWR_NO_SMPS_LPOPEN

In DEEPSTOP, the SMPS is in PRECHARGE mode with output connected to VDDIO

LL_PWR_SMPS_LPOPEN

In DEEPSTOP, the SMPS is disabled with floating output

SMPS step down converter output voltage scaling voltage level

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V20

SMPS step down converter supply output voltage 1.20V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V25

SMPS step down converter supply output voltage 1.25V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V30

SMPS step down converter supply output voltage 1.30V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V35

SMPS step down converter supply output voltage 1.35V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V40

SMPS step down converter supply output voltage 1.40V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V45

SMPS step down converter supply output voltage 1.45V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V50

SMPS step down converter supply output voltage 1.50V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V55

SMPS step down converter supply output voltage 1.55V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V60

SMPS step down converter supply output voltage 1.60V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V65

SMPS step down converter supply output voltage 1.65V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V70

SMPS step down converter supply output voltage 1.70V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V75

SMPS step down converter supply output voltage 1.75V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V80

SMPS step down converter supply output voltage 1.80V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V85

SMPS step down converter supply output voltage 1.85V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V90

SMPS step down converter supply output voltage 1.90V

LL_PWR_SMPS_OUTPUT_VOLTAGE_1V95

SMPS step down converter supply output voltage 1.95V

Select SMPS PRECHARGE limit current

LL_PWR_SMPS_PRECH_LIMIT_CUR_2_5

LL_PWR_SMPS_PRECH_LIMIT_CUR_5

LL_PWR_SMPS_PRECH_LIMIT_CUR_10

LL_PWR_SMPS_PRECH_LIMIT_CUR_20

SMPS PRECHARGE Mode

LL_PWR_NO_SMPS_PRECHARGE

SMPS not in pecharge mode

LL_PWR_SMPS_PRECHARGE

SMPS is disabled and bypassed in precharge mode

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

IO Port for wakeup source

LL_PWR_EWS_PORTA

IO port A to get out from DEEPSTOP mode

LL_PWR_EWS_PORTB

IO port B to get out from DEEPSTOP mode

Wakeup Source

LL_PWR_WAKEUP_CR6_ALL

Enable all the wakeup source in CR6

LL_PWR_WAKEUP_CR3_ALL

Enable all the wakeup source in CR3

LL_PWR_WAKEUP_ALL

LL_PWR_WAKEUP_RTC

Enable wakeup on Internal event (RTC)

LL_PWR_WAKEUP_LPUART

Enable wakeup on Internal event 2 (LPUART)

LL_PWR_WAKEUP_BLEHOST

Enable wakeup on BLE Host CPU event

LL_PWR_WAKEUP_BLE

Enable wakeup on BLE event

LL_PWR_WAKEUP_PA11

Enable wakeup on PA11 I/O event

LL_PWR_WAKEUP_PA10

Enable wakeup on PA10 I/O event

LL_PWR_WAKEUP_PA9

Enable wakeup on PA9 I/O event

LL_PWR_WAKEUP_PA8

Enable wakeup on PA8 I/O event

LL_PWR_WAKEUP_PA3

Enable wakeup on PA3 I/O event

LL_PWR_WAKEUP_PA2

Enable wakeup on PA2 I/O event

LL_PWR_WAKEUP_PA1

Enable wakeup on PA1 I/O event

LL_PWR_WAKEUP_PA0

Enable wakeup on PA0 I/O event

LL_PWR_WAKEUP_PB7

Enable wakeup on PB7 I/O event

LL_PWR_WAKEUP_PB6

Enable wakeup on PB6 I/O event

LL_PWR_WAKEUP_PB5

Enable wakeup on PB5 I/O event

LL_PWR_WAKEUP_PB4

Enable wakeup on PB4 I/O event

LL_PWR_WAKEUP_PB3

Enable wakeup on PB3 I/O event

LL_PWR_WAKEUP_PB2

Enable wakeup on PB2 I/O event

LL_PWR_WAKEUP_PB1

Enable wakeup on PB1 I/O event

LL_PWR_WAKEUP_PB0

Enable wakeup on PB0 I/O event

LL_PWR_WAKEUP_PB15

Enable wakeup on PB15 I/O event

LL_PWR_WAKEUP_PB14

Enable wakeup on PB14 I/O event

LL_PWR_WAKEUP_PB13

Enable wakeup on PB13 I/O event

LL_PWR_WAKEUP_PB12

Enable wakeup on PB12 I/O event

SMPS BOM

PWR_SMPS_BOM1

SMPS BOM1 1.5 uH

PWR_SMPS_BOM2

SMPS BOM2 2.2 uH

PWR_SMPS_BOM3

SMPS BOM3 10 uH

SMPS PRECHARGE Mode

PWR_NO_SMPS_PRECHARGE

SMPS not in pecharge mode

PWR_SMPS_PRECHARGE

SMPS is disabled and bypassed in precharge mode

57 LL RADIO Generic Driver

57.1 RADIO Firmware driver registers structures

57.1.1 GLOBALSTATMACH_WORD_TypeDef

GLOBALSTATMACH_WORD_TypeDef is defined in the stm32wb0x_ll_radio.h

Data Fields

- ***uint32_t WORD0***
- ***uint32_t WORD1***
- ***uint32_t WORD2***
- ***uint32_t WORD3***
- ***uint32_t WORD4***
- ***uint32_t WORD5***
- ***uint32_t WORD6***

Field Documentation

- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD0***
- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD1***
- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD2***
- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD3***
- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD4***
- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD5***
- ***volatile uint32_t GLOBALSTATMACH_WORD_TypeDef::WORD6***

57.2 RADIO Firmware driver defines

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

57.2.1 RADIO

RADIO

58 LL RADIO__TIMER Generic Driver

58.1 RADIO__TIMER Firmware driver API description

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

58.1.1 Detailed description of functions

LL_RADIO_TIMER_EnableTimer1

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableTimer1 (BLUE_TypeDef * BLUEx)

Function description

Enable Radio Timer1.

Parameters

- **BLUEx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMEOUTDESTREG DESTINATION LL_RADIO_TIMER_EnableTimer1

LL_RADIO_TIMER_DisableTimer1

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableTimer1 (BLUE_TypeDef * BLUEx)

Function description

Disable Radio Timer1.

Parameters

- **BLUEx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMEOUTDESTREG DESTINATION LL_RADIO_TIMER_DisableTimer1

LL_RADIO_TIMER_IsEnabledTimer1

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledTimer1 (BLUE_TypeDef * BLUEx)

Function description

Indicates if Radio Timer1 is enabled.

Parameters

- **BLUEx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- TIMEOUTDESTREG DESTINATION LL_RADIO_TIMER_IsEnabledTimer1

LL_RADIO_TIMER_EnableTimer2

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableTimer2 (BLUE_TypeDef * BLUEx)

Function description

Enable Radio Timer2.

Parameters

- **BLUEx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMEOUTDESTREG DESTINATION LL_RADIO_TIMER_EnableTimer2

LL_RADIO_TIMER_DisableTimer2

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableTimer2 (BLUE_TypeDef * BLUEx)

Function description

Disable Radio Timer2.

Parameters

- **BLUEx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- TIMEOUTDESTREG DESTINATION LL_RADIO_TIMER_DisableTimer2

LL_RADIO_TIMER_IsEnabledTimer2

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledTimer2 (BLUE_TypeDef * BLUEx)

Function description

Indicates if Radio Timer2 is enabled.

Parameters

- **BLUEx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- TIMEOUTDESTREG DESTINATION LL_RADIO_TIMER_IsEnabledTimer2

LL_RADIO_TIMER_SetTimeout

Function name

__STATIC_INLINE void LL_RADIO_TIMER_SetTimeout (BLUE_TypeDef * BLUEx, uint32_t timeout)

Function description

Set the Radio Timer Timeout.

Parameters

- **BLUEx:** Radio Timer instance
- **timeout:**

Return values

- **None:**

LL_RADIO_TIMER_GetTimeout

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetTimeout (BLUE_TypeDef * BLUEx)

Function description

Get the Radio Timer Timeout.

Parameters

- **BLUEx:** Radio Timer instance

Return values

- **Timeout:** configured

LL_RADIO_TIMER_GetTimerCapture

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetTimerCapture (BLUE_TypeDef * BLUEx)

Function description

Get the Radio Timer value from the beginning of a new Bluetooth LE sequence.

Parameters

- **BLUEx:** Radio Timer instance

Return values

- **Time:** from the beginning of a new BLE sequence

LL_RADIO_TIMER_EnableBLEWakeupTimer

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableBLEWakeupTimer (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable Radio BLE Wakeup Timer.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE BLE_WAKEUP_EN LL_RADIO_TIMER_EnableBLEWakeupTimer

LL_RADIO_TIMER_DisableBLEWakeupTimer

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableBLEWakeupTimer (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable Radio BLE Wakeup Timer.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE BLE_WAKEUP_EN LL_RADIO_TIMER_DisableBLEWakeupTimer

LL_RADIO_TIMER_IsEnabledBLEWakeupTimer

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledBLEWakeupTimer (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if BLE Wakeup Timer is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE BLE_WAKEUP_EN LL_RADIO_TIMER_IsEnabledBLEWakeupTimer

LL_RADIO_TIMER_EnableCPUWakeupTimer

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableCPUWakeupTimer (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable Radio CPU Wakeup Timer.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE CPU_WAKEUP_EN LL_RADIO_TIMER_EnableCPUWakeupTimer

LL_RADIO_TIMER_DisableCPUWakeupTimer

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableCPUWakeupTimer (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable Radio CPU Wakeup Timer.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE CPU_WAKEUP_EN LL_RADIO_TIMER_DisableCPUWakeupTimer

LL_RADIO_TIMER_IsEnabledCPUWakeupTimer

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledCPUWakeupTimer (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if CPU Wakeup Timer is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE CPU_WAKEUP_EN LL_RADIO_TIMER_IsEnabledCPUWakeupTimer

LL_RADIO_TIMER_SetWakeupOffset

Function name

__STATIC_INLINE void LL_RADIO_TIMER_SetWakeupOffset (WAKEUP_TypeDef * WAKEUPx, uint8_t Time)

Function description

Set the Wakeup Offset to let time to power and clocks to settle (Granularity slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance
- **Time:** Time in slow clock period

Return values

- **None:**

LL_RADIO_TIMER_GetWakeupOffset

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetWakeupOffset (WAKEUP_TypeDef * WAKEUPx)

Function description

Get the Wakeup Offset (Granularity slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **Wakeup:** Time Offset (Granulary slow clock period, size 8 bit)

LL_RADIO_TIMER_SetBLEWakeupTime

Function name

__STATIC_INLINE void LL_RADIO_TIMER_SetBLEWakeupTime (WAKEUP_TypeDef * WAKEUPx, uint32_t Time)

Function description

Set the BLE Wakeup Time (Granularity unit is in 16 x slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance
- **Time:** in 16 x slow clock period

Return values

- **None:**

LL_RADIO_TIMER_GetBLEWakeupTime

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetBLEWakeupTime (WAKEUP_TypeDef * WAKEUPx)

Function description

Get the BLE Wakeup Time (Granularity unit is 16 x slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **BLE:** Wakeup Time

LL_RADIO_TIMER_SetCPUWakeupTime

Function name

__STATIC_INLINE void LL_RADIO_TIMER_SetCPUWakeupTime (WAKEUP_TypeDef * WAKEUPx, uint32_t Time)

Function description

Set the CPU Wakeup Time (Granularity unit is slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance
- **Time:** in slow clock period

Return values

- **None:**

LL_RADIO_TIMER_GetCPUWakeupTime

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetCPUWakeupTime (WAKEUP_TypeDef * WAKEUPx)

Function description

Get the CPU Wakeup Time (Granularity unit is slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **CPU:** Wakeup Time

LL_RADIO_TIMER_GetAbsoluteTime

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetAbsoluteTime (WAKEUP_TypeDef * WAKEUPx)

Function description

Get the Absolute Time of the wakeup timer (Granularity unit is 16 x slow clock period).

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **Absolute:** Time of Wakeup Timer

LL_RADIO_TIMER_EnableWakeupTimerLowPowerMode

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableWakeupTimerLowPowerMode (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable the Low Power Mode for the Wakeup Radio Timers.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE SLEEP_EN LL_RADIO_TIMER_EnableWakeupTimerLowPowerMode

LL_RADIO_TIMER_DisableWakeupTimerLowPowerMode

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableWakeupTimerLowPowerMode (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable the Low Power Mode for the Wakeup Radio Timers.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE SLEEP_EN LL_RADIO_TIMER_DisableWakeupTimerLowPowerMode

LL_RADIO_TIMER_IsEnabledWakeupTimerLowPowerMode

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledWakeupTimerLowPowerMode (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if Wakeup Radio Timer Low Power Mode is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE SLEEP_EN LL_RADIO_TIMER_IsEnabledWakeupTimerLowPowerMode

LL_RADIO_TIMER_EnableBLEWakeupTimerForceSleeping

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableBLEWakeupTimerForceSleeping (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable the SoC to go in low power mode always whatever is the radio status.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE FORCE_SLEEPING LL_RADIO_TIMER_EnableBLEWakeupTimerForceSleeping

LL_RADIO_TIMER_DisableBLEWakeupTimerForceSleeping

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableBLEWakeupTimerForceSleeping (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable the SoC to go in low power mode always.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE FORCE_SLEEPING LL_RADIO_TIMER_DisableBLEWakeupTimerForceSleeping

LL_RADIO_TIMER_IsEnabledBLEWakeupTimerForceSleeping

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledBLEWakeupTimerForceSleeping (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if BLUE Wakeup Radio Timer Force Sleeping Mode is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- BLUE_SLEEP_REQUEST_MODE FORCE_SLEEPING LL_RADIO_TIMER_IsEnabledBLEWakeupTimerForceSleeping

LL_RADIO_TIMER_EnableCPUWakeupTimerForceSleeping

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableCPUWakeupTimerForceSleeping (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable the CPU to be considered always as sleeping by the wakeup block.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE FORCE_SLEEPING LL_RADIO_TIMER_EnableCPUWakeupTimerForceSleeping

LL_RADIO_TIMER_DisableCPUWakeupTimerForceSleeping

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableCPUWakeupTimerForceSleeping (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable the CPU to be considered always as sleeping by the wakeup block.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE FORCE_SLEEPING LL_RADIO_TIMER_DisableCPUWakeupTimerForceSleeping

LL_RADIO_TIMER_IsEnabledCPUWakeupTimerForceSleeping

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledCPUWakeupTimerForceSleeping (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if CPU Wakeup Radio Timer Force Sleeping Mode is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE FORCE_SLEEPING LL_RADIO_TIMER_IsEnabledCPUWakeupTimerForceSleeping

LL_RADIO_TIMER_SetSleepRequestMode

Function name

__STATIC_INLINE void LL_RADIO_TIMER_SetSleepRequestMode (WAKEUP_TypeDef * WAKEUPx, uint8_t mode)

Function description

Set the sleep request mode.

Parameters

- **WAKEUPx:** Radio Timer instance
- **mode:** Sleep mode value in range 0-7

Return values

- **None.:**

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE SLEEP_REQ_MODE LL_RADIO_TIMER_SetSleepRequestMode

LL_RADIO_TIMER_GetSleepRequestMode

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetSleepRequestMode (WAKEUP_TypeDef * WAKEUPx)

Function description

Get the sleep request mode.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **Sleep:** mode value in range 0-7

Reference Manual to LL API cross reference:

- CM0_SLEEP_REQUEST_MODE SLEEP_REQ_MODE LL_RADIO_TIMER_GetSleepRequestMode

LL_RADIO_TIMER_EnableBLEWakeupIT

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableBLEWakeupIT (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable the BLE wakeup interrupt towards the CPU.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- WAKEUP_BLE_IRQ_ENABLE WAKEUP_IT LL_RADIO_TIMER_EnableBLEWakeupIT

LL_RADIO_TIMER_DisableBLEWakeupIT

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableBLEWakeupIT (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable the BLE wakeup interrupt towards the CPU.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- WAKEUP_BLE_IRQ_ENABLE WAKEUP_IT LL_RADIO_TIMER_DisableBLEWakeupIT

LL_RADIO_TIMER_IsEnabledBLEWakeupIT

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledBLEWakeupIT (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if BLE wakeup interrupt towards the CPU is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- WAKEUP_BLE_IRQ_ENABLE WAKEUP_IT LL_RADIO_TIMER_IsEnabledBLEWakeupIT

LL_RADIO_TIMER_EnableCPUWakeupIT

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableCPUWakeupIT (WAKEUP_TypeDef * WAKEUPx)

Function description

Enable the CPU wakeup interrupt towards the CPU.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- WAKEUP_CM0_IRQ_ENABLE WAKEUP_IT LL_RADIO_TIMER_EnableCPUWakeupIT

LL_RADIO_TIMER_DisableCPUWakeupIT

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableCPUWakeupIT (WAKEUP_TypeDef * WAKEUPx)

Function description

Disable the CPU wakeup interrupt towards the CPU.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- WAKEUP_CM0_IRQ_ENABLE WAKEUP_IT LL_RADIO_TIMER_DisableCPUWakeupIT

LL_RADIO_TIMER_IsEnabledCPUWakeupIT

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledCPUWakeupIT (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicates if CPU wakeup interrupt towards the CPU is enabled.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- WAKEUP_CPU_IRQ_ENABLE WAKEUP_IT LL_RADIO_TIMER_IsEnabledCPUWakeupIT

LL_RADIO_TIMER_IsActiveFlag_TIMECAPTURETRIG

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsActiveFlag_TIMECAPTURETRIG (BLUE_TypeDef * BLEx)

Function description

Check if the Radio Timer Time Capture Triggert Flag is set or not.

Parameters

- **BLEx:** BLE Instance

Return values

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

Reference Manual to LL API cross reference:

- STATUSREG TIMECAPTURETRIG LL_RADIO_TIMER_IsActiveFlag_TIMECAPTURETRIG

LL_RADIO_TIMER_ClearFlag_BLEWakeup

Function name

__STATIC_INLINE void LL_RADIO_TIMER_ClearFlag_BLEWakeup (WAKEUP_TypeDef * WAKEUPx)

Function description

Clear the BLE Wakeup interrupt flag.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- WAKEUP_BLE_IRQ_STATUS WAKEUP_IT LL_RADIO_TIMER_ClearFlag_BLEWakeup

LL_RADIO_TIMER_IsActiveFlag_BLEWakeup

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsActiveFlag_BLEWakeup (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicate whether the BLE Wakeup interrupt flag is set.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- WAKEUP_BLE_IRQ_STATUS WAKEUP_IT LL_RADIO_TIMER_IsActiveFlag_BLEWakeup

LL_RADIO_TIMER_ClearFlag_CPUWakeup

Function name

__STATIC_INLINE void LL_RADIO_TIMER_ClearFlag_CPUWakeup (WAKEUP_TypeDef * WAKEUPx)

Function description

Clear the CPU Wakeup interrupt flag.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- WAKEUP_CM0_IRQ_STATUS WAKEUP_IT LL_RADIO_TIMER_ClearFlag_CPUWakeup

LL_RADIO_TIMER_IsActiveFlag_CPUWakeup

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsActiveFlag_CPUWakeup (WAKEUP_TypeDef * WAKEUPx)

Function description

Indicate whether the CPU Wakeup interrupt flag is set.

Parameters

- **WAKEUPx:** Radio Timer instance

Return values

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

Reference Manual to LL API cross reference:

- WAKEUP_CM0_IRQ_STATUS WAKEUP_IT LL_RADIO_TIMER_IsActiveFlag_CPUWakeup

LL_RADIO_TIMER_SetLSIWindowCalibrationLength

Function name

__STATIC_INLINE void LL_RADIO_TIMER_SetLSIWindowCalibrationLength (RADIO_CTRL_TypeDef * RADIO_CTRLx, uint32_t Count)

Function description

Set the window length (in slow clock period) for slow clock measurement.

Parameters

- **RADIO_CTRLx:** Radio Control instance
- **Count:** Slow clock is measured in a window of SLOW_COUNT+1 slow clock cycles

Return values

- **None:**

LL_RADIO_TIMER_GetLSIWindowCalibrationLength

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetLSIWindowCalibrationLength (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Get the window length (in slow clock period) for slow clock measurement.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **The:** LSI Window Length used for the LSI calibration procedure

LL_RADIO_TIMER_StartLSICalibration

Function name

__STATIC_INLINE void LL_RADIO_TIMER_StartLSICalibration (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Start the LSI calibration procedure.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **None:**

LL_RADIO_TIMER_GetLSIPeriod

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetLSIPeriod (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Get the LSI clock Period calibrated.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **The:** LSI clock period calibrated

LL_RADIO_TIMER_GetLSIFrequency

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_GetLSIFrequency (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Get the LSI clock Frequency calibrated.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **The:** LSI clock frequency calibrated

LL_RADIO_TIMER_EnableLSICalibrationIT

Function name

__STATIC_INLINE void LL_RADIO_TIMER_EnableLSICalibrationIT (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Enable the LSI calibration ended interrupt.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **None:**

LL_RADIO_TIMER_DisableLSICalibrationEndedIT

Function name

__STATIC_INLINE void LL_RADIO_TIMER_DisableLSICalibrationEndedIT (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Disable the LSI calibration ended interrupt.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **None:**

LL_RADIO_TIMER_IsEnabledLSICalibrationEndedIT

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsEnabledLSICalibrationEndedIT (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Indicates if the LSI calibration ended interrupt is enabled.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

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

LL_RADIO_TIMER_ClearFlag_LSICalibrationEnded

Function name

__STATIC_INLINE void LL_RADIO_TIMER_ClearFlag_LSICalibrationEnded (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Clear the LSI calibration ended flag.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

- **None:**

LL_RADIO_TIMER_IsActiveFlag_LSICalibrationEnded

Function name

__STATIC_INLINE uint32_t LL_RADIO_TIMER_IsActiveFlag_LSICalibrationEnded (RADIO_CTRL_TypeDef * RADIO_CTRLx)

Function description

Check if the LSI calibration ended flag is set or not.

Parameters

- **RADIO_CTRLx:** Radio Control instance

Return values

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

58.2 RADIO__TIMER Firmware driver defines

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

58.2.1 RADIO__TIMER

RADIO__TIMER

Radio & Wakeup Timer definitions

LL_RADIO_BLE_WAKEUP_TIMER

LL_RADIO_CPU_WAKEUP_TIMER

LL_RADIO_TIMER_1

LL_RADIO_TIMER_2

Flag Defines

LL_RADIO_TIMECAPTURETRIGF

Interrupt Defines

LL_RADIO_BLE_WAKEUP_IT

LL_RADIO_CPU_WAKEUP_IT

59 LL RCC Generic Driver

59.1 RCC Firmware driver registers structures

59.1.1 LL_RCC_ClocksTypeDef

LL_RCC_ClocksTypeDef is defined in the stm32wb0x_ll_rcc.h

Data Fields

- ***uint32_t SYSCLK_Frequency***

Field Documentation

- ***uint32_t LL_RCC_ClocksTypeDef::SYSCLK_Frequency***
  SYSCLK clock frequency

59.2 RCC Firmware driver API description

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

59.2.1 Detailed description of functions

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_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsEnabled (void)

Function description

Check if HSE crystal oscillator is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_HSE_IsEnabled

LL_RCC_HSE_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_IsReady (void)

Function description

Check if HSE oscillator Ready.

Return values

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

Reference Manual to LL API cross reference:

- CR HSERDY LL_RCC_HSE_IsReady

LL_RCC_HSE_SetCapacitorTuning

Function name

__STATIC_INLINE void LL_RCC_HSE_SetCapacitorTuning (uint32_t Value)

Function description

Set HSE capacitor tuning.

Parameters

- **Value:** Between Min_Data = 0 and Max_Data = 63

Return values

- **None:**

Reference Manual to LL API cross reference:

- RFSWHSECR SWXOTUNE LL_RCC_HSE_SetCapacitorTuning

LL_RCC_HSE_GetCapacitorTuning

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_GetCapacitorTuning (void)

Function description

Get HSE capacitor tuning.

Return values

- **Between:** Min_Data = 0 and Max_Data = 63

Reference Manual to LL API cross reference:

- RFSWHSECR SWXOTUNE LL_RCC_HSE_GetCapacitorTuning

LL_RCC_HSE_SetCurrentControl

Function name

__STATIC_INLINE void LL_RCC_HSE_SetCurrentControl (uint32_t CurrentMax)

Function description

Set HSE current control.

Parameters

- **CurrentMax:** This parameter can be one of the following values:
  - LL_RCC_HSE_CURRENTMAX_0
  - LL_RCC_HSE_CURRENTMAX_1
  - LL_RCC_HSE_CURRENTMAX_2
  - LL_RCC_HSE_CURRENTMAX_3
  - LL_RCC_HSE_CURRENTMAX_4
  - LL_RCC_HSE_CURRENTMAX_5
  - LL_RCC_HSE_CURRENTMAX_6
  - LL_RCC_HSE_CURRENTMAX_7

Reference Manual to LL API cross reference:

- RFSWHSECR GMC LL_RCC_HSE_SetCurrentControl

LL_RCC_HSE_GetCurrentControl

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_GetCurrentControl (void)

Function description

Get HSE current control.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_HSE_CURRENTMAX_0
  - LL_RCC_HSE_CURRENTMAX_1
  - LL_RCC_HSE_CURRENTMAX_2
  - LL_RCC_HSE_CURRENTMAX_3
  - LL_RCC_HSE_CURRENTMAX_4
  - LL_RCC_HSE_CURRENTMAX_5
  - LL_RCC_HSE_CURRENTMAX_6
  - LL_RCC_HSE_CURRENTMAX_7

Reference Manual to LL API cross reference:

- RFSWHSECR GMC LL_RCC_HSE_GetCurrentControl

LL_RCC_HSE_SetSenseAmplifier

Function name

__STATIC_INLINE void LL_RCC_HSE_SetSenseAmplifier (uint32_t SenseAmplifier)

Function description

Set HSE sense amplifier threshold.

Parameters

- **SenseAmplifier:** This parameter can be one of the following values:
  - LL_RCC_HSEAMPTHRESHOLD_1_2
  - LL_RCC_HSEAMPTHRESHOLD_3_4

Reference Manual to LL API cross reference:

- RFSWHSECR SATRG LL_RCC_HSE_SetSenseAmplifier

LL_RCC_HSE_GetSenseAmplifier

Function name

__STATIC_INLINE uint32_t LL_RCC_HSE_GetSenseAmplifier (void)

Function description

Get HSE current control.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_HSEAMPTHRESHOLD_1_2
  - LL_RCC_HSEAMPTHRESHOLD_3_4

Reference Manual to LL API cross reference:

- RFSWHSECR SATRG LL_RCC_HSE_GetSenseAmplifier

LL_RCC_HSI_Enable

Function name

__STATIC_INLINE void LL_RCC_HSI_Enable (void)

Function description

Enable HSI.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR STOPHSI LL_RCC_HSI_Enable

LL_RCC_HSI_Disable

Function name

__STATIC_INLINE void LL_RCC_HSI_Disable (void)

Function description

Disable HSI.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR STOPHSI 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_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
- Default value is loaded by HW at reset as soon as the flash controller achieves the reading of the information in flash memory (ICSCR register).

Reference Manual to LL API cross reference:

- CSSWCR 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/CSSWCR HSITRIM LL_RCC_HSI_GetCalibTrimming

LL_RCC_DIRECT_HSE_Enable

Function name

__STATIC_INLINE void LL_RCC_DIRECT_HSE_Enable (void)

Function description

Enable DIRECT_HSE mode.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR HSESEL/STOPHSI LL_RCC_HSE_Enable

LL_RCC_DIRECT_HSE_Disable

Function name

__STATIC_INLINE void LL_RCC_DIRECT_HSE_Disable (void)

Function description

Disable DIRECT_HSE.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR HSESEL/STOPHSI LL_RCC_HSE_Disable

LL_RCC_DIRECT_HSE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_DIRECT_HSE_IsEnabled (void)

Function description

Check if DIRECT_HSE mode is ready.

Return values

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

Reference Manual to LL API cross reference:

- CFGR HSESEL/STOPHSI LL_RCC_DIRECT_HSE_IsEnabled

LL_RCC_Get_DIRECT_HSESEL_Status

Function name

__STATIC_INLINE uint32_t LL_RCC_Get_DIRECT_HSESEL_Status (void)

Function description

Get the DIRECT HSE Selection Status.

Return values

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

Reference Manual to LL API cross reference:

- CFGR HSESEL_STATUS LL_RCC_Get_DIRECT_HSESEL_Status

LL_RCC_LOCKDET_NSTOP_SetTime

Function name

__STATIC_INLINE void LL_RCC_LOCKDET_NSTOP_SetTime (uint32_t time)

Function description

Set LOCKDET_NSTOP time.

Parameters

- **time:** This parameter can be one of the following values:
  – LL_RCC_LOCKDET_NSTOP_0
  – LL_RCC_LOCKDET_NSTOP_1
  – LL_RCC_LOCKDET_NSTOP_2
  – LL_RCC_LOCKDET_NSTOP_3
  – LL_RCC_LOCKDET_NSTOP_4
  – LL_RCC_LOCKDET_NSTOP_5
  – LL_RCC_LOCKDET_NSTOP_6
  – LL_RCC_LOCKDET_NSTOP_7

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR LOCKDET_NSTOP LL_RCC_LOCKDET_NSTOP_SetTime

LL_RCC_LOCKDET_NSTOP_GetTime

Function name

__STATIC_INLINE uint32_t LL_RCC_LOCKDET_NSTOP_GetTime (void)

Function description

Get RC64MPLL prescaler.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_LOCKDET_NSTOP_0
  – LL_RCC_LOCKDET_NSTOP_1
  – LL_RCC_LOCKDET_NSTOP_2
  – LL_RCC_LOCKDET_NSTOP_3
  – LL_RCC_LOCKDET_NSTOP_4
  – LL_RCC_LOCKDET_NSTOP_5
  – LL_RCC_LOCKDET_NSTOP_6
  – LL_RCC_LOCKDET_NSTOP_7

Reference Manual to LL API cross reference:

- CR LOCKDET_NSTOP LL_RCC_LOCKDET_NSTOP_GetTime

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_LSE
  – LL_RCC_LSCO_CLKSOURCE_LSI
  – LL_RCC_LSCO_CLKSOURCE_HSI64M_DIV2048

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CLKSLOWSEL 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_LSE
  – LL_RCC_LSCO_CLKSOURCE_LSI
  – LL_RCC_LSCO_CLKSOURCE_HSI64M_DIV2048

Reference Manual to LL API cross reference:

- CFGR CLKSLOWSEL LL_RCC_LSCO_GetSource

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:

- CR 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:

- CR LSEON LL_RCC_LSE_Disable

LL_RCC_LSE_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsEnabled (void)

Function description

Check if Low Speed External (LSE) crystal has been enabled or not.

Return values

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

Reference Manual to LL API cross reference:

- CR LSEON LL_RCC_LSE_IsEnabled

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:

- CR LSERDY LL_RCC_LSE_IsReady

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:

- CR 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:

- CR LSEBYP LL_RCC_LSE_DisableBypass

LL_RCC_LSE_IsBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_LSE_IsBypassEnabled (void)

Function description

Check if LSE bypass configuration is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CR LSEBYP LL_RCC_LSE_IsBypassEnabled

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:

- CSSWCR 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:

- CSSWCR LSEDRV LL_RCC_LSE_GetDriveCapability

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:

- CR 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:

- CR LSION LL_RCC_LSI_Disable

LL_RCC_LSI_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsEnabled (void)

Function description

Check if LSI crystal oscillator is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CR LSION LL_RCC_LSI_IsEnabled

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:

- CR LSIRDY LL_RCC_LSI_IsReady

LL_RCC_LSI_EnableTrimming

Function name

__STATIC_INLINE void LL_RCC_LSI_EnableTrimming (void)

Function description

Enable LSI trimming procedure.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICSCR LSITRIMEN LL_RCC_LSI_EnableTrimming

LL_RCC_LSI_DisableTrimming

Function name

__STATIC_INLINE void LL_RCC_LSI_DisableTrimming (void)

Function description

Disable LSI trimming procedure.

Return values

- **None:**

Reference Manual to LL API cross reference:

- ICSCR LSITRIMEN LL_RCC_LSI_DisableTrimming

LL_RCC_LSI_IsTrimmingEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_IsTrimmingEnabled (void)

Function description

Check if LSI trimming procedure is enabled.

Return values

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

Reference Manual to LL API cross reference:

- ICSCR LSITRIMEN LL_RCC_LSI_IsTrimmingEnabled

LL_RCC_LSI_SetTrimming

Function name

__STATIC_INLINE void LL_RCC_LSI_SetTrimming (uint32_t Value)

Function description

Set LSI trimming value.

Parameters

- **Value:** Between Min_Data = 0 and Max_Data = 15

Return values

- **None:**

Notes

- After enabling the LSI oscillator and the LSI trimming procedure the LSI trimming value is decreased from 15 to 0, till LSITRIMOK raised

Reference Manual to LL API cross reference:

- CSSWCR LSISWBW LL_RCC_LSI_SetTrimming

LL_RCC_LSI_GetTrimming

Function name

__STATIC_INLINE uint32_t LL_RCC_LSI_GetTrimming (void)

Function description

Get LSI trimming value.

Return values

- **Between:** Min_Data = 0 and Max_Data = 15

Reference Manual to LL API cross reference:

- CSSWCR/ICSCR LSITRIM LL_RCC_LSI_GetTrimming

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_HSI
  – LL_RCC_SYS_HSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEON 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_HSI
  – LL_RCC_SYS_HSE

Reference Manual to LL API cross reference:

- CR HSEON LL_RCC_GetSysClkSource

LL_RCC_SetRFClock

Function name

__STATIC_INLINE void LL_RCC_SetRFClock (uint32_t Source)

Function description

Set the RF clock frequency.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_RF_CLK_16M
  – LL_RCC_RF_CLK_32M

Return values

- **None:**

Reference Manual to LL API cross reference:

- APB2ENR CLKBLEDIV LL_RCC_SetRFClockSource

LL_RCC_GetRFClock

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRFClock (void)

Function description

Get the RF clock frequency.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_RF_CLK_16M
  – LL_RCC_RF_CLK_32M

Reference Manual to LL API cross reference:

- APB2ENR CLKBLEDIV LL_RCC_GetRFClockSource

LL_RCC_IsRFUnderReset

Function name

__STATIC_INLINE uint32_t LL_RCC_IsRFUnderReset (void)

Function description

Check if Radio System is reset.

Return values

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

Reference Manual to LL API cross reference:

- APB2RSTR MRBLERST LL_RCC_IsRFUnderReset

LL_RCC_SetSMPSPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetSMPSPrescaler (uint32_t Prescaler)

Function description

Set SMPS prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  – LL_RCC_SMPS_DIV_2
  – LL_RCC_SMPS_DIV_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR SMPSDIV LL_RCC_SetSMPSPrescaler

LL_RCC_GetSMPSPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSMPSPrescaler (void)

Function description

Get SMPS prescaler.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_SMPS_DIV_2
  – LL_RCC_SMPS_DIV_4

Reference Manual to LL API cross reference:

- CFGR SMPSDIV LL_RCC_GetSMPSPrescaler

LL_RCC_SetLPUARTClockSource

Function name

__STATIC_INLINE void LL_RCC_SetLPUARTClockSource (uint32_t Source)

Function description

Set LPUART Clock source.

Parameters

- **Source:** this parameter can be one of the following values:
  - LL_RCC_LPUCLKSEL_CLK16M
  - LL_RCC_LPUCLKSEL_CLKLSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR LPUCLKSEL LL_RCC_SetLPUARTClockSource

LL_RCC_GetLPUARTClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetLPUARTClockSource (void)

Function description

Get LPUART Clock source.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_LPUCLKSEL_CLK16M
  - LL_RCC_LPUCLKSEL_CLKLSE

Reference Manual to LL API cross reference:

- CFGR LPUCLKSEL LL_RCC_GetLPUARTClockSource

LL_RCC_ConfigMCO

Function name

__STATIC_INLINE void LL_RCC_ConfigMCO (uint32_t MCOSource, uint32_t MCOPrescaler)

Function description

Configure MCOx.

Parameters

- **MCOSource:** This parameter can be one of the following values:
  - LL_RCC_MCOSOURCE_NOCLOCK
  - LL_RCC_MCOSOURCE_SYSCLK
  - LL_RCC_MCOSOURCE_HSI
  - LL_RCC_MCOSOURCE_RC64MPLL
  - LL_RCC_MCOSOURCE_HSE
  - LL_RCC_MCOSOURCE_HSI64M_DIV2048
  - LL_RCC_MCOSOURCE_SMPS
  - LL_RCC_MCOSOURCE_ADC
- **MCOPrescaler:** This parameter can be one of the following values:
  - LL_RCC_MCO_DIV_1
  - LL_RCC_MCO_DIV_2
  - LL_RCC_MCO_DIV_4
  - LL_RCC_MCO_DIV_8
  - LL_RCC_MCO_DIV_16
  - LL_RCC_MCO_DIV_32

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR MCOSEL LL_RCC_ConfigMCO
- CFGR CCOPRE LL_RCC_ConfigMCO

LL_RCC_ConfigLSCO

Function name

__STATIC_INLINE void LL_RCC_ConfigLSCO (uint32_t LSCOSource)

Function description

Configure LSCOx.

Parameters

- **LSCOSource:** This parameter can be one of the following values:
  - LL_RCC_LSCOSOURCE_NOCLOCK
  - LL_RCC_LSCOSOURCE_LSI
  - LL_RCC_LSCOSOURCE_LSE

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR LCOSEL LL_RCC_ConfigLSCO

LL_RCC_LSCOinDeepStop_Enable

Function name

__STATIC_INLINE void LL_RCC_LSCOinDeepStop_Enable (void)

Function description

LCO enabled on GPIA_Pin_10 also in deepstop.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR LCOEN LL_RCC_LSCOinDeepStop_Enable

LL_RCC_LSCOinDeepStop_Disable

Function name

__STATIC_INLINE void LL_RCC_LSCOinDeepStop_Disable (void)

Function description

LCO disabled on GPIA_Pin_10 also in deepstop.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR LCOEN LL_RCC_LSCOinDeepStop_Disable

LL_RCC_LSCOinDeepStop_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_LSCOinDeepStop_IsEnabled (void)

Function description

Check if the LCO is enabled on PA10 in deepstop.

Return values

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

Reference Manual to LL API cross reference:

- CFGR LCOEN LL_RCC_LSCOinDeepStop_IsEnabled

LL_RCC_IOBOOST_Enable

Function name

__STATIC_INLINE void LL_RCC_IOBOOST_Enable (void)

Function description

IO BOOSTER enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR IOBOOSTEN LL_RCC_IOBOOST_Enable

LL_RCC_IOBOOST_Disable

Function name

__STATIC_INLINE void LL_RCC_IOBOOST_Disable (void)

Function description

IO BOOSTER disable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR IOBOOSTEN LL_RCC_IOBOOST_Disable

LL_RCC_IOBOOST_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IOBOOST_IsEnabled (void)

Function description

Check if the IO BOOSTER is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CFGR IOBOOSTEN LL_RCC_IOBOOST_IsEnabled

LL_RCC_IOBOOSTCLK_Enable

Function name

__STATIC_INLINE void LL_RCC_IOBOOSTCLK_Enable (void)

Function description

IO BOOSTER clock enable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR IOBOOSTCLKEN LL_RCC_IOBOOSTCLK_Enable

LL_RCC_IOBOOSTCLK_Disable

Function name

__STATIC_INLINE void LL_RCC_IOBOOSTCLK_Disable (void)

Function description

IO BOOSTER clock disable.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR IOBOOSTCLKEN LL_RCC_IOBOOSTCLK_Disable

LL_RCC_IOBOOSTCLK_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_IOBOOSTCLK_IsEnabled (void)

Function description

Check if the IO BOOSTER clock is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CFGR IOBOOSTCLKEN LL_RCC_IOBOOSTCLK_IsEnabled

LL_RCC_SetSPI3I2SClockSource

Function name

__STATIC_INLINE void LL_RCC_SetSPI3I2SClockSource (uint32_t Source)

Function description

Configure SPI3I2S clock source.

Parameters

- **Source:** This parameter can be one of the following values:
  – LL_RCC_SPI3_I2S_CLK16M
  – LL_RCC_SPI3_I2S_CLK32M
  – LL_RCC_SPI3_I2S_CLK64M

Return values

- **None:**

Notes

- The LL_RCC_SPI3_I2S_CLK64M is valid for STM32WB05 and STM32WB09 family

Reference Manual to LL API cross reference:

- CFGR SPI3I2SCLKSEL LL_RCC_SetSPI3I2SClockSource

LL_RCC_GetSPI3I2SClockSource

Function name

__STATIC_INLINE uint32_t LL_RCC_GetSPI3I2SClockSource (void)

Function description

Get SPI3I2S clock source.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_SPI3_I2S_CLK16M
  – LL_RCC_SPI3_I2S_CLK32M
  – LL_RCC_SPI3_I2S_CLK64M

Notes

- The LL_RCC_SPI3_I2S_CLK64M is valid for STM32WB05 and STM32WB09 family

Reference Manual to LL API cross reference:

- CFGR SPI3I2SCLKSEL LL_RCC_GetSPI3I2SClockSource

LL_RCC_RC64MPLL_Enable

Function name

__STATIC_INLINE void LL_RCC_RC64MPLL_Enable (void)

Function description

Enable RC64MPLL.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSIPLLON LL_RCC_RC64MPLL_Enable

LL_RCC_RC64MPLL_Disable

Function name

__STATIC_INLINE void LL_RCC_RC64MPLL_Disable (void)

Function description

Disable RC64MPLL.

Return values

- **None:**

Notes

- Cannot be disabled if the RC64MPLL clock is used as the system clock

Reference Manual to LL API cross reference:

- CR HSIPLLON LL_RCC_RC64MPLL_Disable

LL_RCC_RC64MPLL_IsReady

Function name

__STATIC_INLINE uint32_t LL_RCC_RC64MPLL_IsReady (void)

Function description

Check if RC64MPLL is Ready.

Return values

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

Reference Manual to LL API cross reference:

- CR HSIPLLRDY LL_RCC_RC64MPLL_IsReady

LL_RCC_SetRC64MPLLPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetRC64MPLLPrescaler (uint32_t Prescaler)

Function description

Set RC64MPLL prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  – LL_RCC_RC64MPLL_DIV_1
  – LL_RCC_RC64MPLL_DIV_2
  – LL_RCC_RC64MPLL_DIV_4
  – LL_RCC_RC64MPLL_DIV_8
  – LL_RCC_RC64MPLL_DIV_16
  – LL_RCC_RC64MPLL_DIV_32
  – LL_RCC_RC64MPLL_DIV_64

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CLKSYSDIV LL_RCC_SetRC64MPLLPrescaler

LL_RCC_GetRC64MPLLPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRC64MPLLPrescaler (void)

Function description

Get RC64MPLL prescaler.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_RC64MPLL_DIV_1
  – LL_RCC_RC64MPLL_DIV_2
  – LL_RCC_RC64MPLL_DIV_4
  – LL_RCC_RC64MPLL_DIV_8
  – LL_RCC_RC64MPLL_DIV_16
  – LL_RCC_RC64MPLL_DIV_32
  – LL_RCC_RC64MPLL_DIV_64

Reference Manual to LL API cross reference:

- CFGR CLKSYSDIV LL_RCC_GetRC64MPLLPrescaler

LL_RCC_GetCLKSYSPrescalerStatus

Function name

__STATIC_INLINE uint32_t LL_RCC_GetCLKSYSPrescalerStatus (void)

Function description

Get System Clock Prescaler Status.

Return values

- **Returned:** value can be one of the following values:
  – LL_RCC_RC64MPLL_DIV_1
  – LL_RCC_RC64MPLL_DIV_2
  – LL_RCC_RC64MPLL_DIV_4
  – LL_RCC_RC64MPLL_DIV_8
  – LL_RCC_RC64MPLL_DIV_16
  – LL_RCC_RC64MPLL_DIV_32
  – LL_RCC_RC64MPLL_DIV_64
  – LL_RCC_DIRECT_HSE_DIV_1
  – LL_RCC_DIRECT_HSE_DIV_2
  – LL_RCC_DIRECT_HSE_DIV_4
  – LL_RCC_DIRECT_HSE_DIV_8
  – LL_RCC_DIRECT_HSE_DIV_16
  – LL_RCC_DIRECT_HSE_DIV_32

Reference Manual to LL API cross reference:

- CFGR CLKSYSDIV_STATUS LL_RCC_GetCLKSYSPrescalerStatus

LL_RCC_SetDirectHSEPrescaler

Function name

__STATIC_INLINE void LL_RCC_SetDirectHSEPrescaler (uint32_t Prescaler)

Function description

Set DIRECT_HSE prescaler.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  - LL_RCC_DIRECT_HSE_DIV_1
  - LL_RCC_DIRECT_HSE_DIV_2
  - LL_RCC_DIRECT_HSE_DIV_4
  - LL_RCC_DIRECT_HSE_DIV_8
  - LL_RCC_DIRECT_HSE_DIV_16
  - LL_RCC_DIRECT_HSE_DIV_32

Return values

- **None:**

Reference Manual to LL API cross reference:

- CFGR CLKSYSDIV LL_RCC_SetDirectHSEPrescaler

LL_RCC_GetDirectHSEPrescaler

Function name

__STATIC_INLINE uint32_t LL_RCC_GetDirectHSEPrescaler (void)

Function description

Get DIRECT_HSE prescaler.

Return values

- **Returned:** value can be one of the following values:
  - LL_RCC_DIRECT_HSE_DIV_1
  - LL_RCC_DIRECT_HSE_DIV_2
  - LL_RCC_DIRECT_HSE_DIV_4
  - LL_RCC_DIRECT_HSE_DIV_8
  - LL_RCC_DIRECT_HSE_DIV_16
  - LL_RCC_DIRECT_HSE_DIV_32

Reference Manual to LL API cross reference:

- CFGR CLKSYSDIV LL_RCC_GetDirectHSEPrescaler

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:

- CIFR LSIRDYF 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:

- CIFR LSERDYF 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:

- CIFR HSIRDYF 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:

- CIFR HSERDYF LL_RCC_ClearFlag_HSERDY

LL_RCC_ClearFlag_RC64MPLLRDY

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_RC64MPLLRDY (void)

Function description

Clear RC64MPLL ready interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFR HSIPLLRDYF LL_RCC_ClearFlag_RC64MPLLRDY

LL_RCC_ClearFlag_RC64MPLLUNLOCKDET

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_RC64MPLLUNLOCKDET (void)

Function description

Clear RC64MPLL Unlock Detection interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFR HSIPLLUNLOCKDETF LL_RCC_ClearFlag_RC64MPLLUNLOCKDET

LL_RCC_ClearFlag_RTCRSTREL

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_RTCRSTREL (void)

Function description

Clear RTC Reset Release interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFR RTCRSTF LL_RCC_ClearFlag_RTCRRSTREL

LL_RCC_ClearFlag_WDGRSTREL

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_WDGRSTREL (void)

Function description

Clear Watchdog Reset Release interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFR WDGRSTF LL_RCC_ClearFlag_WDGRSTREL

LL_RCC_ClearFlag_LPURSTREL

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_LPURSTREL (void)

Function description

Clear LPUART Reset Release interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIFR LPURSTF LL_RCC_ClearFlag_LPURSTREL

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_RC64MPLLRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_RC64MPLLRDY (void)

Function description

Check if RC64MPLL ready interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIFR HSIPLLRDYF LL_RCC_IsActiveFlag_RC64MPLLRDY

LL_RCC_IsActiveFlag_RC64MPLLUNLOCKDET

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_RC64MPLLUNLOCKDET (void)

Function description

Check if RC64MPLL Unlock Detection flag interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIFR HSIPLLUNLOCKDETF LL_RCC_IsActiveFlag_RC64MPLLUNLOCKDET

LL_RCC_IsActiveFlag_RTCRSTREL

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_RTCRSTREL (void)

Function description

Check if RTC Reset Release flag interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIFR RTCRSTF LL_RCC_IsActiveFlag_RTCRSTREL

LL_RCC_IsActiveFlag_WDGRSTREL

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WDGRSTREL (void)

Function description

Check if WDG Reset Release flag interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIFR WDGRSTF LL_RCC_IsActiveFlag_WDGRSTREL

LL_RCC_IsActiveFlag_LPURSTREL

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LPURSTREL (void)

Function description

Check if LPUART Reset Release flag interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CIFR LPURSTF LL_RCC_IsActiveFlag_LPURSTREL

LL_RCC_IsActiveFlag_LOCKUPRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_LOCKUPRST (void)

Function description

Check if RCC flag CPU lockup reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR LOCKUPRSTF LL_RCC_IsActiveFlag_LOCKUPRST

LL_RCC_IsActiveFlag_WDGRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_WDGRST (void)

Function description

Check if RCC flag Watchdog reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR WDGRSTF LL_RCC_IsActiveFlag_WDGRST

LL_RCC_IsActiveFlag_SFTRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_SFTRST (void)

Function description

Check if RCC flag Software reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR SFTRSTF LL_RCC_IsActiveFlag_SFTRST

LL_RCC_IsActiveFlag_PORRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PORRST (void)

Function description

Check if RCC flag Power-On reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR PORRSTF LL_RCC_IsActiveFlag_PORRST

LL_RCC_IsActiveFlag_PADRST

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_PADRST (void)

Function description

Check if RCC flag NRSTn pad reset is set or not.

Return values

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

Reference Manual to LL API cross reference:

- CSR PADRSTF LL_RCC_IsActiveFlag_PADRST

LL_RCC_ClearResetFlags

Function name

__STATIC_INLINE void LL_RCC_ClearResetFlags (void)

Function description

Set RMVF bit to clear the reset flags.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSR RMVF LL_RCC_ClearResetFlags

LL_RCC_SwitchRC64MPLLPrescaler

Function name

__STATIC_INLINE void LL_RCC_SwitchRC64MPLLPrescaler (uint32_t Prescaler)

Function description

Set RC64MPLL prescaler to switch the clock when the MR_BLE is enabled.

Parameters

- **Prescaler:** This parameter can be one of the following values:
  – LL_RCC_RC64MPLL_SWITCH_DIV_1
  – LL_RCC_RC64MPLL_SWITCH_DIV_2
  – LL_RCC_RC64MPLL_SWITCH_DIV_4

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSCMDR CLKSYSDIV_REQ LL_RCC_SwitchRC64MPLLPrescaler

LL_RCC_RequestSwitchRC64MPLLClock

Function name

__STATIC_INLINE void LL_RCC_RequestSwitchRC64MPLLClock (void)

Function description

Request the RC64MPLL prescaler to switch the clock when the MR_BLE is enabled.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSCMDR REQUEST LL_RCC_RequestSwitchRC64MPLLClock

LL_RCC_GetRC64MPLLSwitchStatus

Function name

__STATIC_INLINE uint32_t LL_RCC_GetRC64MPLLSwitchStatus (void)

Function description

Get RC64MPLL clock switch status.

Return values

- **The:** Clock switch status. Possible values are:
  – LL_RCC_RC64MPLL_SWITCH_STATUS_IDLE
  – LL_RCC_RC64MPLL_SWITCH_STATUS_ONGOING
  – LL_RCC_RC64MPLL_SWITCH_STATUS_DONE

Reference Manual to LL API cross reference:

- CSCMDR STATUS LL_RCC_HSE_GetCapacitorTuning

LL_RCC_EnableIT_EOFSEQ

Function name

__STATIC_INLINE void LL_RCC_EnableIT_EOFSEQ (void)

Function description

Enable End of Switch Sequence interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSCMDR EOFSEQ_IE LL_RCC_EnableIT_EOFSEQ

LL_RCC_DisableIT_EOFSEQ

Function name

__STATIC_INLINE void LL_RCC_DisableIT_EOFSEQ (void)

Function description

Disable End of Switch Sequence interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSCMDR EOFSEQ_IE LL_RCC_DisableIT_EOFSEQ

LL_RCC_IsEnabledIT_EOFSEQ

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_EOFSEQ (void)

Function description

Checks if End of Switch Sequence interrupt source is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CSCMDR EOFSEQ_IE LL_RCC_IsEnabledIT_EOFSEQ

LL_RCC_ClearFlag_EOFSEQ

Function name

__STATIC_INLINE void LL_RCC_ClearFlag_EOFSEQ (void)

Function description

Clear End of Sequence interrupt flag.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CSCMDR EOFSEQ_IRQ LL_RCC_ClearFlag_EOFSEQ

LL_RCC_IsActiveFlag_EOFSEQ

Function name

__STATIC_INLINE uint32_t LL_RCC_IsActiveFlag_EOFSEQ (void)

Function description

Check if End of Sequence interrupt occurred or not.

Return values

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

Reference Manual to LL API cross reference:

- CSCMDR EOFSEQ_IRQ LL_RCC_IsActiveFlag_EOFSEQ

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_EnableIT_RC64MPLLRDY

Function name

__STATIC_INLINE void LL_RCC_EnableIT_RC64MPLLRDY (void)

Function description

Enable RC64MPLL ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIPLLRDYIE LL_RCC_EnableIT_RC64MPLLRDY

LL_RCC_EnableIT_RC64MPLLUNLOCKDET

Function name

__STATIC_INLINE void LL_RCC_EnableIT_RC64MPLLUNLOCKDET (void)

Function description

Enable RC64MPLL Unlock Detection interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_HSIPLLUNLOCKDETIE LL_RCC_EnableIT_RC64MPLLUNLOCKDET

LL_RCC_EnableIT_RTCRSTREL

Function name

__STATIC_INLINE void LL_RCC_EnableIT_RTCRSTREL (void)

Function description

Enable RTC Reset Release interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_RTCRSTIE LL_RCC_EnableIT_RTCRSTREL

LL_RCC_EnableIT_WDGRSTREL

Function name

__STATIC_INLINE void LL_RCC_EnableIT_WDGRSTREL (void)

Function description

Enable Watchdog Reset Release interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_WDGRSTIE LL_RCC_EnableIT_WDGRSTREL

LL_RCC_EnableIT_LPURSTREL

Function name

__STATIC_INLINE void LL_RCC_EnableIT_LPURSTREL (void)

Function description

Enable LPUART Reset Release interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_LPURSTIE LL_RCC_EnableIT_LPURSTREL

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_DisableIT_RC64MPLLRDY

Function name

__STATIC_INLINE void LL_RCC_DisableIT_RC64MPLLRDY (void)

Function description

Disable RC64MPLL ready interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER HSIPLLRDYIE LL_RCC_DisableIT_RC64MPLLRDY

LL_RCC_DisableIT_RC64MPLLUNLOCKDET

Function name

__STATIC_INLINE void LL_RCC_DisableIT_RC64MPLLUNLOCKDET (void)

Function description

Disable RC64MPLL Unlock Detection interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_HSIPLLUNLOCKDETIE LL_RCC_DisableIT_RC64MPLLUNLOCKDET

LL_RCC_DisableIT_RTCRSTREL

Function name

__STATIC_INLINE void LL_RCC_DisableIT_RTCRSTREL (void)

Function description

Disable RTC Reset Release interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_RTCRSTIE LL_RCC_DisableIT_RTCRSTREL

LL_RCC_DisableIT_WDGRSTREL

Function name

__STATIC_INLINE void LL_RCC_DisableIT_WDGRSTREL (void)

Function description

Disable Watchdog Reset Release interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_WDGRSTIE LL_RCC_DisableIT_WDGRSTREL

LL_RCC_DisableIT_LPURSTREL

Function name

__STATIC_INLINE void LL_RCC_DisableIT_LPURSTREL (void)

Function description

Disable LPUART Reset Release interrupt.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_LPURSTIE LL_RCC_DisableIT_LPURSTREL

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_IsEnabledIT_RC64MPLLRDY

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_RC64MPLLRDY (void)

Function description

Checks if RC64MPLL ready interrupt source is enabled or disabled.

Return values

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

Reference Manual to LL API cross reference:

- CIER HSIPLLRDYIE LL_RCC_IsEnabledIT_RC64MPLLRDY

LL_RCC_IsEnabledIT_RC64MPLLUNLOCKDET

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_RC64MPLLUNLOCKDET (void)

Function description

Checks if RC64MPLL Unlock Detection interrupt source is enabled or disabled.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_HSIPLLUNLOCKDETIE LL_RCC_IsEnabledIT_RC64MPLLUNLOCKDET

LL_RCC_IsEnabledIT_RTCRSTREL

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_RTCRSTREL (void)

Function description

Checks if RTC Reset Release interrupt source is enabled or disabled.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_RTCRSTIE LL_RCC_IsEnabledIT_RTCRSTREL

LL_RCC_IsEnabledIT_WDGRSTREL

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_WDGRSTREL (void)

Function description

Checks if Watchdog Reset Release interrupt source is enabled or disabled.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_WDGRSTIE LL_RCC_IsEnabledIT_WDGRSTREL

LL_RCC_IsEnabledIT_LPURSTREL

Function name

__STATIC_INLINE uint32_t LL_RCC_IsEnabledIT_LPURSTREL (void)

Function description

Checks if LPUART Reset Release interrupt source is enabled or disabled.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CIER RCC_CIER_LPURSTIE LL_RCC_IsEnabledIT_LPURSTREL

LL_RCC_HSEPLLBUFON_Enable

Function name

__STATIC_INLINE void LL_RCC_HSEPLLBUFON_Enable (void)

Function description

Enable high speed clock buffer for PLL RF2G4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEPLLBUFON LL_RCC_HSEPLLBUFON_Enable

LL_RCC_HSEPLLBUFON_Disable

Function name

__STATIC_INLINE void LL_RCC_HSEPLLBUFON_Disable (void)

Function description

Disable high speed clock buffer for PLL RF2G4.

Return values

- **None:**

Reference Manual to LL API cross reference:

- CR HSEPLLBUFON LL_RCC_HSEPLLBUFON_Disable

LL_RCC_HSEPLLBUFON_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RCC_HSEPLLBUFON_IsEnabled (void)

Function description

Check if the high speed clock buffer for PLL RF2G4 is enabled.

Return values

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

Reference Manual to LL API cross reference:

- CR HSEPLLBUFON LL_RCC_HSEPLLBUFON_IsEnabled

LL_RCC_DeInit

Function name

ErrorStatus LL_RCC_DeInit (void)

Function description

Reset the RCC clock 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 and PLL Source OFFAll interrupts disabled
- This function doesn't modify the configuration of the Peripheral clocksLSI, LSE

LL_RCC_GetSystemClocksFreq

Function name

void LL_RCC_GetSystemClocksFreq (LL_RCC_ClocksTypeDef * RCC_Clocks)

Function description

Return the frequency of the system clock.

Parameters

- **RCC_Clocks:** pointer to a LL_RCC_ClocksTypeDef structure which will hold the clocks frequencies

Return values

- **None:**

LL_RCC_GetSMPSClockFreq

Function name

uint32_t LL_RCC_GetSMPSClockFreq (void)

Function description

Return SMPS clock frequency.

Return values

- **SMPS:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that the peripheral has a wrong ANADIV prescaler setup

Notes

- This function is only applicable when CPU runs.

LL_RCC_GetLPUARTClockFreq

Function name

uint32_t LL_RCC_GetLPUARTClockFreq (uint32_t LPUARTxSource)

Function description

Return LPUARTx clock frequency.

Parameters

- **LPUARTxSource:** This parameter can be one of the following values:
  – LL_RCC_LPUART1_CLKSOURCE

Return values

- **LPUART:** clock frequency (in Hz)
  – LL_RCC_PERIPH_FREQUENCY_NO indicates that oscillator (CLK_16M or LSE) is not ready

59.3 RCC Firmware driver defines

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

59.3.1 RCC

RCC

LPUART clock selection

LL_RCC_LPUCLKSEL_CLK16M

LPUART Clock Selection 16 MHz

LL_RCC_LPUCLKSEL_CLKLSE

LPUART Clock Selection LSE source

Ready Interrupt Flags Defines

LL_RCC_CIFR_LSIRDYF

LSI Ready Interrupt Flag/Clear

LL_RCC_CIFR_LSERDYF

LSE Ready Interrupt Flag/Clear

LL_RCC_CIFR_HSIRDYF

HSI Ready Interrupt Flag/Clear

LL_RCC_CIFR_HSERDYF

HSE Ready Interrupt Flag/Clear

LL_RCC_CIFR_PLLRDYF

PLL Ready Interrupt Flag/Clear

LL_RCC_CIFR_PLLUNLOCKDETF

PLL Unlock Interrupt Flag/Clear

LL_RCC_CIFR_RTCRSTRELF

RTC Reset Release Interrupt Flag/Clear

LL_RCC_CIFR_WDGRSTRELF

WDG Reset Release Interrupt Flag/Clear

LL_RCC_CIFR_LPURSTRELF

LPUART Reset Release Interrupt Flag/Clear

Reset Flags Defines

LL_RCC_CSR_LOCKUPRSTF

CPU lockup reset flag

LL_RCC_CSR_WDGRSTF

Watchdog reset flag

LL_RCC_CSR_SFTRSTF

Software reset flag

LL_RCC_CSR_PORRSTF

Power-On or BOR reset flag

LL_RCC_CSR_PADRSTF

NRSTn pad reset flag

HSE current control max limits

LL_RCC_HSE_CURRENTMAX_0

HSE current control max limit = 0.18 mA/V

LL_RCC_HSE_CURRENTMAX_1

HSE current control max limit = 0.57 mA/V

LL_RCC_HSE_CURRENTMAX_2

HSE current control max limit = 0.78 mA/V

LL_RCC_HSE_CURRENTMAX_3

HSE current control max limit = 1.13 mA/V

LL_RCC_HSE_CURRENTMAX_4

HSE current control max limit = 0.61 mA/V

LL_RCC_HSE_CURRENTMAX_5

HSE current control max limit = 1.65 mA/V

LL_RCC_HSE_CURRENTMAX_6

HSE current control max limit = 2.12 mA/V

LL_RCC_HSE_CURRENTMAX_7

HSE current control max limit = 2.84 mA/V

HSE sense amplifier threshold

LL_RCC_HSEAMPTHRESHOLD_1_2

HSE sense amplifier bias current factor = 1/2

LL_RCC_HSEAMPTHRESHOLD_3_4

HSE sense amplifier bias current factor = 3/4

IT Enable Defines

LL_RCC_CIER_PLLRDYIE

PLL ready interrupt enable

LL_RCC_CIER_HSERDYIE

HSE ready interrupt enable

LL_RCC_CIER_HSIRDYIE

HSI ready interrupt enable

LL_RCC_CIER_LSERDYIE

LSE ready interrupt enable

LL_RCC_CIER_LSIRDYIE

LSI ready interrupt enable

LOCKDET_NSTOP time selection

LL_RCC_LOCKDET_NSTOP_0

LL_RCC_LOCKDET_NSTOP_1

LL_RCC_LOCKDET_NSTOP_2

LL_RCC_LOCKDET_NSTOP_3

LL_RCC_LOCKDET_NSTOP_4

LL_RCC_LOCKDET_NSTOP_5

LL_RCC_LOCKDET_NSTOP_6

LL_RCC_LOCKDET_NSTOP_7

LPUART1

LL_RCC_LPUART1_CLKSOURCE

LPUART1 clock source

LCO SOURCE selection

LL_RCC_LSCOSOURCE_NOCLOCK

LSCO output disabled, no clock on LSCO

LL_RCC_LSCOSOURCE_LSI

LSI selection as LSCO source

LL_RCC_LSCOSOURCE_LSE

LSE selection as LSCO source

LSCO Source Selection

LL_RCC_LSCO_CLKSOURCE_LSE

LSE selection for low speed clock

LL_RCC_LSCO_CLKSOURCE_LSI

Enable the LSI selection for low speed clock

LL_RCC_LSCO_CLKSOURCE_HSI64M_DIV2048

HSI_64M divided by 2048 selection for low speed clock

Enable LSCO Clock Source

LL_RCC_LSCO_LSI

Enable LSI selection for low speed clock

LL_RCC_LSCO_LSE

Enable LSE selection for low speed clock

LL_RCC_LSCO_LSEBYP

Enable LSE clock bypass 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

MCO SOURCE selection

LL_RCC_MCOSOURCE_NOCLOCK

MCO output disabled, no clock on MCO

LL_RCC_MCOSOURCE_SYSCLK

SYSCLK selection as MCO source

LL_RCC_MCOSOURCE_HSI

HSI selection as MCO source

LL_RCC_MCOSOURCE_RC64MPLL

RC64MPLL selection as MCO source

LL_RCC_MCOSOURCE_HSE

HSE after stabilization selection as MCO source

LL_RCC_MCOSOURCE_HSI64M_DIV2048

HSI_64M divided by 2048 seelction as MCO source

LL_RCC_MCOSOURCE_SMPS

SMPS selection as MCO source

LL_RCC_MCOSOURCE_ADC

ADC selection as MCO source

MCO prescaler

LL_RCC_MCO_DIV_1

MCO not divided

LL_RCC_MCO_DIV_2

MCO divided by 2

LL_RCC_MCO_DIV_4

MCO divided by 4

LL_RCC_MCO_DIV_8
MCO divided by 8

LL_RCC_MCO_DIV_16
MCO divided by 16

LL_RCC_MCO_DIV_32
MCO divided by 32

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

RF system clock

LL_RCC_RF_CLK_32M
Radio system clock 32 MHz

LL_RCC_RF_CLK_16M
Radio system clock 16 MHz

SMPS prescaler

LL_RCC_SMPS_DIV_2
SMPS clock division 2 (SMPS clock is 8 MHz)

LL_RCC_SMPS_DIV_4
SMPS clock division 4 (SMPS clock is 4 MHz)

SPI3_I2S

LL_RCC_SPI3_I2S_CLK16M
SPI3 I2S 16 MHz clock source selection bits

LL_RCC_SPI3_I2S_CLK32M
SPI3 I2S 32 MHz clock source selection bits

LL_RCC_SPI3_I2S_CLK64M
SPI3 I2S 64 MHz clock source selection bits

RC64MPLL divider factor to have the System clock

LL_RCC_RC64MPLL_DIV_1
RC64MPLL not divided as SYSCLK

LL_RCC_RC64MPLL_DIV_2
RC64MPLL divided by 2 as SYSCLK

LL_RCC_RC64MPLL_DIV_4
RC64MPLL divided by 4 as SYSCLK

LL_RCC_RC64MPLL_DIV_8
RC64MPLL divided by 8 as SYSCLK

LL_RCC_RC64MPLL_DIV_16

RC64MPLL divided by 16 as SYSCLK

LL_RCC_RC64MPLL_DIV_32

RC64MPLL divided by 32 as SYSCLK

LL_RCC_RC64MPLL_DIV_64

RC64MPLL divided by 64 as SYSCLK

RC64MPLL divider factor to switch the System clock with MR_BLE enabled

LL_RCC_RC64MPLL_SWITCH_DIV_1

RC64MPLL not divided as SYSCLK

LL_RCC_RC64MPLL_SWITCH_DIV_2

RC64MPLL divided by 2 as SYSCLK

LL_RCC_RC64MPLL_SWITCH_DIV_4

RC64MPLL divided by 4 as SYSCLK

Status of the RC64MPLL clock switch with MR_BLE enabled

LL_RCC_RC64MPLL_SWITCH_STATUS_IDLE

RC64MPLL Clock Switch Status IDLE

LL_RCC_RC64MPLL_SWITCH_STATUS_ONGOING

RC64MPLL Clock Switch Status ONGOING

LL_RCC_RC64MPLL_SWITCH_STATUS_DONE

RC64MPLL Clock Switch Status DONE

Enable HS clock Source

LL_RCC_SYS_HSI

Enable HSI system clock

LL_RCC_SYS_HSE

Enable HSE system clock

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

Enable the External high speed clock buffer for PLL

LL_RCC_HSEPLLBUF_OFF

Disable the External high speed clock buffer for PLL

LL_RCC_HSEPLLBUF_ON

Enable the External high speed clock buffer for PLL. Always enabled when the radio is used

60 LL RTC Generic Driver

60.1 RTC Firmware driver registers structures

60.1.1 LL_RTC_InitTypeDef

LL_RTC_InitTypeDef is defined in the stm32wb0x_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()**.

60.1.2 LL_RTC_TimeTypeDef

LL_RTC_TimeTypeDef is defined in the stm32wb0x_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()**.

60.1.3 LL_RTC_DateTypeDef

LL_RTC_DateTypeDef is defined in the stm32wb0x_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()**.

60.1.4 LL_RTC_AlarmTypeDef

LL_RTC_AlarmTypeDef is defined in the stm32wb0x_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()**.
- ***uint32_t LL_RTC_AlarmTypeDef::AlarmDateWeekDaySel***
  Specifies the RTC Alarm is on day or WeekDay. This parameter can be a value of ***RTC_LL_EC_ALMA_WEEKDAY_SELECTION*** for ALARM A.This feature can be modified afterwards using unitary function **LL_RTC_ALMA_EnableWeekday()** or **LL_RTC_ALMA_DisableWeekday()**.
- ***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()**.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()**.

60.2 RTC Firmware driver API description

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

60.2.1 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:

- CR FMT LL_RTC_SetHourFormat

LL_RTC_GetHourFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_GetHourFormat (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:

- CR FMT LL_RTC_GetHourFormat

LL_RTC_SetAlarmOutEvent

Function name

__STATIC_INLINE void LL_RTC_SetAlarmOutEvent (RTC_TypeDef * RTCx, uint32_t AlarmOutput)

Function description

Select the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance
- **AlarmOutput:** This parameter can be one of the following values:
  - LL_RTC_ALARMOUT_DISABLE
  - LL_RTC_ALARMOUT_ALMA
  - LL_RTC_ALARMOUT_ALMB
  - LL_RTC_ALARMOUT_WAKEUP

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR OSEL LL_RTC_SetAlarmOutEvent

LL_RTC_GetAlarmOutEvent

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAlarmOutEvent (RTC_TypeDef * RTCx)

Function description

Get the flag to be routed to RTC_ALARM output.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_ALARMOUT_DISABLE
  – LL_RTC_ALARMOUT_ALMA
  – LL_RTC_ALARMOUT_ALMB
  – LL_RTC_ALARMOUT_WAKEUP

Reference Manual to LL API cross reference:

- CR OSEL LL_RTC_GetAlarmOutEvent

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:

- ISR 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:

- ISR 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/ALRBF/WUTF is asserted)

Parameters

- **RTCx:** RTC Instance
- **Polarity:** This parameter can be one of the following values:
  – LL_RTC_OUTPUTPOLARITY_PIN_HIGH
  – LL_RTC_OUTPUTPOLARITY_PIN_LOW

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR POL LL_RTC_SetOutputPolarity

LL_RTC_GetOutputPolarity

Function name

__STATIC_INLINE uint32_t LL_RTC_GetOutputPolarity (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:

- 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:

- 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:

- CR BYPSHAD LL_RTC_DisableShadowRegBypass

LL_RTC_IsShadowRegBypassEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_IsShadowRegBypassEnabled (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:

- CR BYPSHAD LL_RTC_IsShadowRegBypassEnabled

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:

- 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:

- PRER PREDIV_S LL_RTC_SetSynchPrescaler

LL_RTC_GetAsynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetAsynchPrescaler (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:

- PRER PREDIV_A LL_RTC_GetAsynchPrescaler

LL_RTC_GetSynchPrescaler

Function name

__STATIC_INLINE uint32_t LL_RTC_GetSynchPrescaler (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:

- 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:

- 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:

- WPR KEY LL_RTC_DisableWriteProtection

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:

- TR PM LL_RTC_TIME_SetFormat

LL_RTC_TIME_GetFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetFormat (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:

- 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:

- TR HT LL_RTC_TIME_SetHour
- TR HU LL_RTC_TIME_SetHour

LL_RTC_TIME_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetHour (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:

- TR HT LL_RTC_TIME_GetHour
- 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:

- TR MNT LL_RTC_TIME_SetMinute
- TR MNU LL_RTC_TIME_SetMinute

LL_RTC_TIME_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetMinute (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:

- TR MNT LL_RTC_TIME_GetMinute
- 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:

- TR ST LL_RTC_TIME_SetSecond
- TR SU LL_RTC_TIME_SetSecond

LL_RTC_TIME_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSecond (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:

- TR ST LL_RTC_TIME_GetSecond
- 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:

- TR PM LL_RTC_TIME_Config
- TR HT LL_RTC_TIME_Config
- TR HU LL_RTC_TIME_Config
- TR MNT LL_RTC_TIME_Config
- TR MNU LL_RTC_TIME_Config
- TR ST LL_RTC_TIME_Config
- TR SU LL_RTC_TIME_Config

LL_RTC_TIME_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_Get (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:

- TR HT LL_RTC_TIME_Get
- TR HU LL_RTC_TIME_Get
- TR MNT LL_RTC_TIME_Get
- TR MNU LL_RTC_TIME_Get
- TR ST LL_RTC_TIME_Get
- 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:

- 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:

- CR BKP LL_RTC_TIME_DisableDayLightStore

LL_RTC_TIME_IsDayLightStoreEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_IsDayLightStoreEnabled (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:

- 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:

- 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:

- CR ADD1H LL_RTC_TIME_IncHour

LL_RTC_TIME_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_TIME_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get subseconds value in the synchronous prescaler counter.

Parameters

- **RTCx:** RTC Instance

Return values

- **Subseconds:** 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:

- 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:

- SHIFTR ADD1S LL_RTC_TIME_Synchronize
- 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:

- DR YT LL_RTC_DATE_SetYear
- DR YU LL_RTC_DATE_SetYear

LL_RTC_DATE_GetYear

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetYear (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:

- DR YT LL_RTC_DATE_GetYear
- 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:

- DR WDU LL_RTC_DATE_SetWeekDay

LL_RTC_DATE_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetWeekDay (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:

- 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:

- DR MT LL_RTC_DATE_SetMonth
- DR MU LL_RTC_DATE_SetMonth

LL_RTC_DATE_GetMonth

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetMonth (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:

- DR MT LL_RTC_DATE_GetMonth
- 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:

- DR DT LL_RTC_DATE_SetDay
- DR DU LL_RTC_DATE_SetDay

LL_RTC_DATE_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_GetDay (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:

- DR DT LL_RTC_DATE_GetDay
- 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:

- DR WDU LL_RTC_DATE_Config
- DR MT LL_RTC_DATE_Config
- DR MU LL_RTC_DATE_Config
- DR DT LL_RTC_DATE_Config
- DR DU LL_RTC_DATE_Config
- DR YT LL_RTC_DATE_Config
- DR YU LL_RTC_DATE_Config

LL_RTC_DATE_Get

Function name

__STATIC_INLINE uint32_t LL_RTC_DATE_Get (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:

- DR WDU LL_RTC_DATE_Get
- DR MT LL_RTC_DATE_Get
- DR MU LL_RTC_DATE_Get
- DR DT LL_RTC_DATE_Get
- DR DU LL_RTC_DATE_Get
- DR YT LL_RTC_DATE_Get
- 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:

- 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:

- 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:

- ALRMAR MSK4 LL_RTC_ALMA_SetMask
- ALRMAR MSK3 LL_RTC_ALMA_SetMask
- ALRMAR MSK2 LL_RTC_ALMA_SetMask
- ALRMAR MSK1 LL_RTC_ALMA_SetMask

LL_RTC_ALMA_GetMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMask (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:

- ALRMAR MSK4 LL_RTC_ALMA_GetMask
- ALRMAR MSK3 LL_RTC_ALMA_GetMask
- ALRMAR MSK2 LL_RTC_ALMA_GetMask
- 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:

- 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:

- 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:

- ALRMAR DT LL_RTC_ALMA_SetDay
- ALRMAR DU LL_RTC_ALMA_SetDay

LL_RTC_ALMA_GetDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetDay (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:

- ALRMAR DT LL_RTC_ALMA_GetDay
- 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:

- ALRMAR DU LL_RTC_ALMA_SetWeekDay

LL_RTC_ALMA_GetWeekDay

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetWeekDay (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:

- 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:

- ALRMAR PM LL_RTC_ALMA_SetTimeFormat

LL_RTC_ALMA_GetTimeFormat

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetTimeFormat (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:

- 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:

- ALRMAR HT LL_RTC_ALMA_SetHour
- ALRMAR HU LL_RTC_ALMA_SetHour

LL_RTC_ALMA_GetHour

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetHour (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:

- ALRMAR HT LL_RTC_ALMA_GetHour
- 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:

- ALRMAR MNT LL_RTC_ALMA_SetMinute
- ALRMAR MNU LL_RTC_ALMA_SetMinute

LL_RTC_ALMA_GetMinute

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetMinute (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:

- ALRMAR MNT LL_RTC_ALMA_GetMinute
- 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:

- ALRMAR ST LL_RTC_ALMA_SetSecond
- ALRMAR SU LL_RTC_ALMA_SetSecond

LL_RTC_ALMA_GetSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSecond (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:

- ALRMAR ST LL_RTC_ALMA_GetSecond
- 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:

- ALRMAR PM LL_RTC_ALMA_ConfigTime
- ALRMAR HT LL_RTC_ALMA_ConfigTime
- ALRMAR HU LL_RTC_ALMA_ConfigTime
- ALRMAR MNT LL_RTC_ALMA_ConfigTime
- ALRMAR MNU LL_RTC_ALMA_ConfigTime
- ALRMAR ST LL_RTC_ALMA_ConfigTime
- 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 B Time (hour, minute and second) in BCD format.

Parameters

- **RTCx:** RTC Instance

Return values

- **Combination:** of hours, minutes and seconds.

Notes

- helper macros __LL_RTC_GET_HOUR, __LL_RTC_GET_MINUTE and __LL_RTC_GET_SECOND are available to get independently each parameter.

Reference Manual to LL API cross reference:

- ALRMAR HT LL_RTC_ALMA_GetTime
- ALRMAR HU LL_RTC_ALMA_GetTime
- ALRMAR MNT LL_RTC_ALMA_GetTime
- ALRMAR MNU LL_RTC_ALMA_GetTime
- ALRMAR ST LL_RTC_ALMA_GetTime
- 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

Mask the most-significant bits of the subseconds field starting from the bit specified in parameter Mask.

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:

- ALRMASSR MASKSS LL_RTC_ALMA_SetSubSecondMask

LL_RTC_ALMA_GetSubSecondMask

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecondMask (RTC_TypeDef * RTCx)

Function description

Get Alarm A subseconds mask.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- 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 subseconds 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:

- ALRMASSR SS LL_RTC_ALMA_SetSubSecond

LL_RTC_ALMA_GetSubSecond

Function name

__STATIC_INLINE uint32_t LL_RTC_ALMA_GetSubSecond (RTC_TypeDef * RTCx)

Function description

Get Alarm A subseconds value.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data=0x7FFF

Reference Manual to LL API cross reference:

- ALRMASSR SS LL_RTC_ALMA_GetSubSecond

LL_RTC_WAKEUP_Enable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Enable (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR WUTE LL_RTC_WAKEUP_Enable

LL_RTC_WAKEUP_Disable

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_Disable (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR WUTE LL_RTC_WAKEUP_Disable

LL_RTC_WAKEUP_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_IsEnabled (RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- CR WUTE LL_RTC_WAKEUP_IsEnabled

LL_RTC_WAKEUP_SetClock

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetClock (RTC_TypeDef * RTCx, uint32_t WakeupClock)

Function description

Select Wakeup clock.

Parameters

- **RTCx:** RTC Instance
- **WakeupClock:** This parameter can be one of the following values:
  – LL_RTC_WAKEUPCLOCK_DIV_16
  – LL_RTC_WAKEUPCLOCK_DIV_8
  – LL_RTC_WAKEUPCLOCK_DIV_4
  – LL_RTC_WAKEUPCLOCK_DIV_2
  – LL_RTC_WAKEUPCLOCK_CKSPRE
  – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RTC_CR WUTE bit = 0 and RTC_ISR WUTWF bit = 1

Reference Manual to LL API cross reference:

- CR WUCKSEL LL_RTC_WAKEUP_SetClock

LL_RTC_WAKEUP_GetClock

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetClock (RTC_TypeDef * RTCx)

Function description

Get Wakeup clock.

Parameters

- **RTCx:** RTC Instance

Return values

- **Returned:** value can be one of the following values:
  – LL_RTC_WAKEUPCLOCK_DIV_16
  – LL_RTC_WAKEUPCLOCK_DIV_8
  – LL_RTC_WAKEUPCLOCK_DIV_4
  – LL_RTC_WAKEUPCLOCK_DIV_2
  – LL_RTC_WAKEUPCLOCK_CKSPRE
  – LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

Reference Manual to LL API cross reference:

- CR WUCKSEL LL_RTC_WAKEUP_GetClock

LL_RTC_WAKEUP_SetAutoReload

Function name

__STATIC_INLINE void LL_RTC_WAKEUP_SetAutoReload (RTC_TypeDef * RTCx, uint32_t Value)

Function description

Set Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance
- **Value:** Value between Min_Data=0x00 and Max_Data=0xFFFF

Return values

- **None:**

Notes

- Bit can be written only when WUTWF is set to 1 in RTC_ISR

Reference Manual to LL API cross reference:

- WUTR WUT LL_RTC_WAKEUP_SetAutoReload

LL_RTC_WAKEUP_GetAutoReload

Function name

__STATIC_INLINE uint32_t LL_RTC_WAKEUP_GetAutoReload (RTC_TypeDef * RTCx)

Function description

Get Wakeup auto-reload value.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- WUTR WUT LL_RTC_WAKEUP_GetAutoReload

LL_RTC_BAK_SetRegister

Function name

__STATIC_INLINE void LL_RTC_BAK_SetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister, uint32_t Data)

Function description

Writes a data in a specified RTC Backup data register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4
- **Data:** Value between Min_Data=0x00 and Max_Data=0xFFFFFFFF

Return values

- **None:**

Reference Manual to LL API cross reference:

- BKPxR BKP LL_RTC_BAK_SetRegister

LL_RTC_BAK_GetRegister

Function name

__STATIC_INLINE uint32_t LL_RTC_BAK_GetRegister (RTC_TypeDef * RTCx, uint32_t BackupRegister)

Function description

Reads data from the specified RTC Backup data Register.

Parameters

- **RTCx:** RTC Instance
- **BackupRegister:** This parameter can be one of the following values:
  - LL_RTC_BKP_DR0
  - LL_RTC_BKP_DR1
  - LL_RTC_BKP_DR2
  - LL_RTC_BKP_DR3
  - LL_RTC_BKP_DR4

Return values

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

Reference Manual to LL API cross reference:

- BKPxR BKP LL_RTC_BAK_GetRegister

LL_RTC_CAL_SetOutputFreq

Function name

__STATIC_INLINE void LL_RTC_CAL_SetOutputFreq (RTC_TypeDef * RTCx, uint32_t Frequency)

Function description

Set Calibration output frequency (1 Hz or 512 Hz)

Parameters

- **RTCx:** RTC Instance
- **Frequency:** This parameter can be one of the following values:
  - LL_RTC_CALIB_OUTPUT_NONE
  - LL_RTC_CALIB_OUTPUT_1HZ
  - LL_RTC_CALIB_OUTPUT_512HZ

Return values

- **None:**

Notes

- Bits are write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR COE LL_RTC_CAL_SetOutputFreq
- CR COSEL LL_RTC_CAL_SetOutputFreq

LL_RTC_CAL_GetOutputFreq

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetOutputFreq (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:

- CR COE LL_RTC_CAL_GetOutputFreq
- CR COSEL LL_RTC_CAL_GetOutputFreq

LL_RTC_CAL_SetPulse

Function name

__STATIC_INLINE void LL_RTC_CAL_SetPulse (RTC_TypeDef * RTCx, uint32_t Pulse)

Function description

Insert or not One RTCCLK pulse every 2exp11 pulses (frequency increased by 488.5 ppm)

Parameters

- **RTCx:** RTC Instance
- **Pulse:** This parameter can be one of the following values:
  - LL_RTC_CALIB_INSERTPULSE_NONE
  - LL_RTC_CALIB_INSERTPULSE_SET

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:

- CALR CALP LL_RTC_CAL_SetPulse

LL_RTC_CAL_IsPulseInserted

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_IsPulseInserted (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:

- 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 smooth calibration cycle period.

Parameters

- **RTCx:** RTC Instance
- **Period:** This parameter can be one of the following values:
  - LL_RTC_CALIB_PERIOD_32SEC
  - LL_RTC_CALIB_PERIOD_16SEC
  - LL_RTC_CALIB_PERIOD_8SEC

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:

- CALR CALW8 LL_RTC_CAL_SetPeriod
- CALR CALW16 LL_RTC_CAL_SetPeriod

LL_RTC_CAL_GetPeriod

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetPeriod (RTC_TypeDef * RTCx)

Function description

Get smooth 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:

- CALR CALW8 LL_RTC_CAL_GetPeriod
- 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 smooth Calibration minus.

Parameters

- **RTCx:** RTC Instance
- **CalibMinus:** Value between Min_Data=0x00 and Max_Data=0x1FF

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.
- Bit can be written only when RECALPF is set to 0 in RTC_ISR

Reference Manual to LL API cross reference:

- CALR CALM LL_RTC_CAL_SetMinus

LL_RTC_CAL_GetMinus

Function name

__STATIC_INLINE uint32_t LL_RTC_CAL_GetMinus (RTC_TypeDef * RTCx)

Function description

Get smooth Calibration minus.

Parameters

- **RTCx:** RTC Instance

Return values

- **Value:** between Min_Data=0x00 and Max_Data= 0x1FF

Reference Manual to LL API cross reference:

- CALR CALM LL_RTC_CAL_GetMinus

LL_RTC_IsActiveFlag_RECALP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RECALP (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:

- ISR RECALPF LL_RTC_IsActiveFlag_RECALP

LL_RTC_IsActiveFlag_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUT (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR WUTF LL_RTC_IsActiveFlag_WUT

LL_RTC_IsActiveFlag_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRA (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:

- ISR ALRAF LL_RTC_IsActiveFlag_ALRA

LL_RTC_ClearFlag_WUT

Function name

__STATIC_INLINE void LL_RTC_ClearFlag_WUT (RTC_TypeDef * RTCx)

Function description

Clear Wakeup timer flag.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Reference Manual to LL API cross reference:

- ISR WUTF LL_RTC_ClearFlag_WUT

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:

- ISR ALRAF LL_RTC_ClearFlag_ALRA

LL_RTC_IsActiveFlag_INIT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INIT (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:

- ISR INITF LL_RTC_IsActiveFlag_INIT

LL_RTC_IsActiveFlag_RS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_RS (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:

- ISR 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:

- ISR RSF LL_RTC_ClearFlag_RS

LL_RTC_IsActiveFlag_INITS

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_INITS (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:

- ISR INITS LL_RTC_IsActiveFlag_INITS

LL_RTC_IsActiveFlag_SHP

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_SHP (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:

- ISR SHPF LL_RTC_IsActiveFlag_SHP

LL_RTC_IsActiveFlag_WUTW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_WUTW (RTC_TypeDef * RTCx)

Function description

Get Wakeup timer write flag.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- ISR WUTWF LL_RTC_IsActiveFlag_WUTW

LL_RTC_IsActiveFlag_ALRAW

Function name

__STATIC_INLINE uint32_t LL_RTC_IsActiveFlag_ALRAW (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:

- ISR ALRAWF LL_RTC_IsActiveFlag_ALRAW

LL_RTC_EnableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_EnableIT_WUT (RTC_TypeDef * RTCx)

Function description

Enable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR WUTIE LL_RTC_EnableIT_WUT

LL_RTC_DisableIT_WUT

Function name

__STATIC_INLINE void LL_RTC_DisableIT_WUT (RTC_TypeDef * RTCx)

Function description

Disable Wakeup timer interrupt.

Parameters

- **RTCx:** RTC Instance

Return values

- **None:**

Notes

- Bit is write-protected. LL_RTC_DisableWriteProtection function should be called before.

Reference Manual to LL API cross reference:

- CR WUTIE LL_RTC_DisableIT_WUT

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:

- 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:

- CR ALRAIE LL_RTC_DisableIT_ALRA

LL_RTC_IsEnabledIT_WUT

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_WUT (RTC_TypeDef * RTCx)

Function description

Check if Wakeup timer interrupt is enabled or not.

Parameters

- **RTCx:** RTC Instance

Return values

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

Reference Manual to LL API cross reference:

- CR WUTIE LL_RTC_IsEnabledIT_WUT

LL_RTC_IsEnabledIT_ALRA

Function name

__STATIC_INLINE uint32_t LL_RTC_IsEnabledIT_ALRA (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:

- 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.

60.3 RTC Firmware driver defines

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

60.3.1 RTC

RTC

ALARM OUTPUT

LL_RTC_ALARMOUT_DISABLE

Output disabled

LL_RTC_ALARMOUT_ALMA

Alarm A output enabled

LL_RTC_ALARMOUT_ALMB

Alarm B output enabled

LL_RTC_ALARMOUT_WAKEUP

Wakeup output enabled

ALARM OUTPUT TYPE

LL_RTC_ALARM_OUTPUTTYPE_OPENDRAIN

RTC_ALARM, when mapped on PC13, is open-drain output

LL_RTC_ALARM_OUTPUTTYPE_PUSHPULL

RTC_ALARM, when mapped on PC13, 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

BACKUP

LL_RTC_BKP_DR0

LL_RTC_BKP_DR1

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_ISR_RECALPF

LL_RTC_ISR_TSOVF

LL_RTC_ISR_TSF

LL_RTC_ISR_WUTF

LL_RTC_ISR_ALRAF

LL_RTC_ISR_INITF

LL_RTC_ISR_RSF

LL_RTC_ISR_INITS

LL_RTC_ISR_SHPF

LL_RTC_ISR_WUTWF

LL_RTC_ISR_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_WUTIE

LL_RTC_CR_ALRAIE

LL_RTC__TAMPIE

MONTH

LL_RTC_MONTH_JANUARY

January

LL_RTC_MONTH_FEBRUARY

February

LL_RTC_MONTH_MARCH
March

LL_RTC_MONTH_APRIL
April

LL_RTC_MONTH_MAY
May

LL_RTC_MONTH_JUNE
June

LL_RTC_MONTH_JULY
July

LL_RTC_MONTH_AUGUST
August

LL_RTC_MONTH_SEPTEMBER
September

LL_RTC_MONTH_OCTOBER
October

LL_RTC_MONTH_NOVEMBER
November

LL_RTC_MONTH_DECEMBER
December

OUTPUT POLARITY PIN

LL_RTC_OUTPUTPOLARITY_PIN_HIGH
Pin is high when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

LL_RTC_OUTPUTPOLARITY_PIN_LOW
Pin is low when ALRAF/ALRBF/WUTF is asserted (depending on OSEL)

SHIFT SECOND

LL_RTC_SHIFT_SECOND_DELAY

LL_RTC_SHIFT_SECOND_ADVANCE

TIME FORMAT

LL_RTC_TIME_FORMAT_AM_OR_24
AM or 24-hour format

LL_RTC_TIME_FORMAT_PM
PM

WAKEUP CLOCK DIV

LL_RTC_WAKEUPCLOCK_DIV_16
RTC/16 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_8

RTC/8 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_4

RTC/4 clock is selected

LL_RTC_WAKEUPCLOCK_DIV_2

RTC/2 clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE

ck_spre (usually 1 Hz) clock is selected

LL_RTC_WAKEUPCLOCK_CKSPRE_WUT

ck_spre (usually 1 Hz) clock is selected and 2exp16 is added to the WUT counter value

WEEK DAY

LL_RTC_WEEKDAY_MONDAY

Monday

LL_RTC_WEEKDAY_TUESDAY

Tuesday

LL_RTC_WEEKDAY_WEDNESDAY

Wednesday

LL_RTC_WEEKDAY_THURSDAY

Thrusday

LL_RTC_WEEKDAY_FRIDAY

Friday

LL_RTC_WEEKDAY_SATURDAY

Saturday

LL_RTC_WEEKDAY_SUNDAY

Sunday

Convert helper Macros

__LL_RTC_CONVERT_BIN2BCD

Description:

- Helper macro to convert a value from 2 digit decimal format to BCD format.

Parameters:

- __VALUE__: Byte to be converted

Return value:

- Converted: byte

__LL_RTC_CONVERT_BCD2BIN

Description:

- Helper macro to convert a value from BCD format to 2 digit decimal format.

Parameters:

- __VALUE__: BCD value to be converted

Return value:

- Converted: byte

Date helper Macros

__LL_RTC_GET_WEEKDAY

Description:

- Helper macro to retrieve weekday.

Parameters:

- __RTC_DATE__: Date returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_WEEKDAY_MONDAY
  - LL_RTC_WEEKDAY_TUESDAY
  - LL_RTC_WEEKDAY_WEDNESDAY
  - LL_RTC_WEEKDAY_THURSDAY
  - LL_RTC_WEEKDAY_FRIDAY
  - LL_RTC_WEEKDAY_SATURDAY
  - LL_RTC_WEEKDAY_SUNDAY

__LL_RTC_GET_YEAR

Description:

- Helper macro to retrieve Year in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Year: in BCD format (0x00 . . . 0x99)

__LL_RTC_GET_MONTH

Description:

- Helper macro to retrieve Month in BCD format.

Parameters:

- __RTC_DATE__: Value returned by

Return value:

- Returned: value can be one of the following values:
  - LL_RTC_MONTH_JANUARY
  - LL_RTC_MONTH_FEBRUARY
  - LL_RTC_MONTH_MARCH
  - LL_RTC_MONTH_APRIL
  - LL_RTC_MONTH_MAY
  - LL_RTC_MONTH_JUNE
  - LL_RTC_MONTH_JULY
  - LL_RTC_MONTH_AUGUST
  - LL_RTC_MONTH_SEPTEMBER
  - LL_RTC_MONTH_OCTOBER
  - LL_RTC_MONTH_NOVEMBER
  - LL_RTC_MONTH_DECEMBER

__LL_RTC_GET_DAY

Description:
- Helper macro to retrieve Day in BCD format.

Parameters:
- __RTC_DATE__: Value returned by

Return value:
- Day: in BCD format (0x01 . . . 0x31)

Time helper Macros

__LL_RTC_GET_HOUR

Description:
- Helper macro to retrieve hour in BCD format.

Parameters:
- __RTC_TIME__: RTC time returned by

Return value:
- Hours: in BCD format (0x01. . .0x12 or between Min_Data=0x00 and Max_Data=0x23)

__LL_RTC_GET_MINUTE

Description:
- Helper macro to retrieve minute in BCD format.

Parameters:
- __RTC_TIME__: RTC time returned by

Return value:
- Minutes: in BCD format (0x00. . .0x59)

__LL_RTC_GET_SECOND

Description:
- Helper macro to retrieve second in BCD format.

Parameters:
- __RTC_TIME__: RTC time returned by

Return value:
- Seconds: in format (0x00. . .0x59)

Common Write and read registers Macros

LL_RTC_WriteReg

Description:
- Write a value in RTC register.

Parameters:
- __INSTANCE__: RTC Instance
- __REG__: Register to be written
- __VALUE__: Value to be written in the register

Return value:
- None

LL_RTC_ReadReg

Description:

- Read a value in RTC register.

Parameters:

- __INSTANCE__: RTC Instance
- __REG__: Register to be read

Return value:

- Register: value

61 LL SPI Generic Driver

61.1 SPI Firmware driver registers structures

61.1.1 LL_SPI_InitTypeDef

LL_SPI_InitTypeDef is defined in the stm32wb0x_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()**.

61.2 SPI Firmware driver API description

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

61.2.1 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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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 (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:**

61.3 SPI Firmware driver defines

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

61.3.1 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

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

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

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

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

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

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

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_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

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

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

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

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

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

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)

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

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

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

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

62 LL SYSTEM Generic Driver

62.1 SYSTEM Firmware driver registers structures

62.1.1 LL_SYSCFG_IO_InitTypeDef

LL_SYSCFG_IO_InitTypeDef is defined in the stm32wb0x_ll_system.h

Data Fields

- ***uint32_t Line***
- ***FunctionalState LineCommand***
- ***uint8_t Type***
- ***uint8_t Trigger***

Field Documentation

- ***uint32_t LL_SYSCFG_IO_InitTypeDef::Line***
  Specifies the EXTI lines to be enabled or disabled for Lines in range PA0 to PB15 This parameter can be any combination of EXTI_LL_EC_LINE
- ***FunctionalState LL_SYSCFG_IO_InitTypeDef::LineCommand***
  Specifies the new state of the selected EXTI lines. This parameter can be set either to ENABLE or DISABLE
- ***uint8_t LL_SYSCFG_IO_InitTypeDef::Type***
  Specifies the type for the EXTI lines EDGE or LEVEL. This parameter can be a value of EXTI_LL_EC_TYPE.
- ***uint8_t LL_SYSCFG_IO_InitTypeDef::Trigger***
  Specifies the trigger signal active edge/level for the EXTI lines. This parameter can be a value of EXTI_LL_EC_TRIGGER.

62.2 SYSTEM Firmware driver API description

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

62.2.1 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

62.2.2 Detailed description of functions

LL_SYSCFG_GetDeviceID

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetDeviceID (void)

Function description

Get the device version and cut information.

Return values

- **Device:** ID

Reference Manual to LL API cross reference:

- DIE_ID PRODUCT/VERSION/REVISION LL_SYSCFG_GetDeviceID

LL_SYSCFG_GetDeviceRevision

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetDeviceRevision (void)

Function description

Get the device cut revision (metal fix) information.

Return values

- **Device:** ID

Reference Manual to LL API cross reference:

- DIE_ID REVISION LL_SYSCFG_GetDeviceRevision

LL_SYSCFG_GetDeviceVersion

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetDeviceVersion (void)

Function description

Get the device cut version information.

Return values

- **Device:** ID

Reference Manual to LL API cross reference:

- DIE_ID VERSION LL_SYSCFG_GetDeviceVersion

LL_SYSCFG_GetDeviceJTAG_ID

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_GetDeviceJTAG_ID (void)

Function description

Get the JTAG_ID device information.

Return values

- **Device:** ID

Reference Manual to LL API cross reference:

- JTAG_ID LL_SYSCFG_GetDeviceJTAG_ID

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_SRAM

Return values

- **None:**

Reference Manual to LL API cross reference:

- CONFIG REMAP 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_SRAM

Reference Manual to LL API cross reference:

- CONFIG REMAP LL_SYSCFG_GetRemapMemory

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_PA0
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA1
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB10
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB11
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA7
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA13
  - LL_SYSCFG_I2C_FASTMODEPLUS_PA14
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  - LL_SYSCFG_I2C_FASTMODEPLUS_PB7

Return values

- **None:**

Notes

- LL_SYSCFG_I2C_FASTMODEPLUS_PB10, LL_SYSCFG_I2C_FASTMODEPLUS_PB11, LL_SYSCFG_I2C_FASTMODEPLUS_PA6, LL_SYSCFG_I2C_FASTMODEPLUS_PA7, LL_SYSCFG_I2C_FASTMODEPLUS_PA13 and LL_SYSCFG_I2C_FASTMODEPLUS_PA14

Reference Manual to LL API cross reference:

- I2C_FMP_CTRL I2C1_PAx_FMP/I2C2_PBx_FMP 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_PA0
  – LL_SYSCFG_I2C_FASTMODEPLUS_PA1
  – LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  – LL_SYSCFG_I2C_FASTMODEPLUS_PB7
  – LL_SYSCFG_I2C_FASTMODEPLUS_PB10
  – LL_SYSCFG_I2C_FASTMODEPLUS_PB11
  – LL_SYSCFG_I2C_FASTMODEPLUS_PA6
  – LL_SYSCFG_I2C_FASTMODEPLUS_PA7
  – LL_SYSCFG_I2C_FASTMODEPLUS_PA13
  – LL_SYSCFG_I2C_FASTMODEPLUS_PA14
  – LL_SYSCFG_I2C_FASTMODEPLUS_PB6
  – LL_SYSCFG_I2C_FASTMODEPLUS_PB7

Return values

- **None:**

Notes

- LL_SYSCFG_I2C_FASTMODEPLUS_PB10, LL_SYSCFG_I2C_FASTMODEPLUS_PB11, LL_SYSCFG_I2C_FASTMODEPLUS_PA6, LL_SYSCFG_I2C_FASTMODEPLUS_PA7, LL_SYSCFG_I2C_FASTMODEPLUS_PA13 and LL_SYSCFG_I2C_FASTMODEPLUS_PA14

Reference Manual to LL API cross reference:

- I2C_FMP_CTRL I2C1_PAx_FMP/I2C2_PBx_FMP LL_SYSCFG_DisableFastModePlus

LL_SYSCFG_PWRC_EnableIT

Function name

__STATIC_INLINE void LL_SYSCFG_PWRC_EnableIT (uint32_t Interrupt)

Function description

Enable PWRC Interrupt Mask.

Parameters

- **Interrupt:** This parameter can be a combination of the following values:
  – LL_SYSCFG_PWRC_PVD
  – LL_SYSCFG_PWRC_WKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- PWRC_IER LL_SYSCFG_PWRC_EnableIT

LL_SYSCFG_PWRC_DisableIT

Function name

__STATIC_INLINE void LL_SYSCFG_PWRC_DisableIT (uint32_t Interrupt)

Function description

Disable PWRC Interrupt Mask.

Parameters

- **Interrupt:** This parameter can be a combination of the following values:
  – LL_SYSCFG_PWRC_PVD
  – LL_SYSCFG_PWRC_WKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- PWRC_IER LL_SYSCFG_PWRC_DisableIT

LL_SYSCFG_PWRC_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_PWRC_IsEnabledIT (uint32_t Interrupt)

Function description

Indicate if PWRC Interrupt Mask is enabled.

Parameters

- **Interrupt:** This parameter can be a combination of the following values:
  – LL_SYSCFG_PWRC_PVD
  – LL_SYSCFG_PWRC_WKUP

Return values

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

Reference Manual to LL API cross reference:

- PWRC_IER LL_SYSCFG_PWRC_IsEnabledIT

LL_SYSCFG_PWRC_GetStatusIT

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_PWRC_GetStatusIT (uint32_t Interrupt)

Function description

Get PWRC Interrupt Status.

Parameters

- **Interrupt:** This parameter can be a combination of the following values:
  – LL_SYSCFG_PWRC_PVD
  – LL_SYSCFG_PWRC_WKUP

Return values

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

Reference Manual to LL API cross reference:

- PWRC_ISCR LL_SYSCFG_PWRC_GetStatusIT

LL_SYSCFG_PWRC_ClearIT

Function name

__STATIC_INLINE void LL_SYSCFG_PWRC_ClearIT (uint32_t Interrupt)

Function description

Clear PWRC Interrupt Status.

Parameters

- **Interrupt:** This parameter can be a combination of the following values:
  – LL_SYSCFG_PWRC_PVD
  – LL_SYSCFG_PWRC_WKUP

Return values

- **None:**

Reference Manual to LL API cross reference:

- PWRC_ISCR LL_SYSCFG_PWRC_ClearIT

LL_EXTI_EnableIT

Function name

__STATIC_INLINE void LL_EXTI_EnableIT (uint32_t ExtiLine)

Function description

Enable IO Interrupt for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_PA0
  – LL_EXTI_LINE_PA1
  – LL_EXTI_LINE_PA2
  – LL_EXTI_LINE_PA3
  – LL_EXTI_LINE_PA4
  – LL_EXTI_LINE_PA5
  – LL_EXTI_LINE_PA6
  – LL_EXTI_LINE_PA7
  – LL_EXTI_LINE_PA8
  – LL_EXTI_LINE_PA9
  – LL_EXTI_LINE_PA10
  – LL_EXTI_LINE_PA11
  – LL_EXTI_LINE_PA12
  – LL_EXTI_LINE_PA13
  – LL_EXTI_LINE_PA14
  – LL_EXTI_LINE_PA15
  – LL_EXTI_LINE_PB0
  – LL_EXTI_LINE_PB1
  – LL_EXTI_LINE_PB2
  – LL_EXTI_LINE_PB3
  – LL_EXTI_LINE_PB4
  – LL_EXTI_LINE_PB5
  – LL_EXTI_LINE_PB6
  – LL_EXTI_LINE_PB7
  – LL_EXTI_LINE_PB8
  – LL_EXTI_LINE_PB9
  – LL_EXTI_LINE_PB10
  – LL_EXTI_LINE_PB11
  – LL_EXTI_LINE_PB12
  – LL_EXTI_LINE_PB13
  – LL_EXTI_LINE_PB14
  – LL_EXTI_LINE_PB15
  – LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_IER LL_EXTI_EnableIT

LL_EXTI_DisableIT

Function name

__STATIC_INLINE void LL_EXTI_DisableIT (uint32_t ExtiLine)

Function description

Disable IO Interrupt for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_IER LL_EXTI_DisableIT

LL_EXTI_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledIT (uint32_t ExtiLine)

Function description

Indicate if IO Interrupt Mask is enabled for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

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

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_IER LL_EXTI_IsEnabledIT

LL_EXTI_EnableEdgeDetection

Function name

__STATIC_INLINE void LL_EXTI_EnableEdgeDetection (uint32_t ExtiLine)

Function description

Set IO Interrupt Type for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_PA0
  – LL_EXTI_LINE_PA1
  – LL_EXTI_LINE_PA2
  – LL_EXTI_LINE_PA3
  – LL_EXTI_LINE_PA4
  – LL_EXTI_LINE_PA5
  – LL_EXTI_LINE_PA6
  – LL_EXTI_LINE_PA7
  – LL_EXTI_LINE_PA8
  – LL_EXTI_LINE_PA9
  – LL_EXTI_LINE_PA10
  – LL_EXTI_LINE_PA11
  – LL_EXTI_LINE_PA12
  – LL_EXTI_LINE_PA13
  – LL_EXTI_LINE_PA14
  – LL_EXTI_LINE_PA15
  – LL_EXTI_LINE_PB0
  – LL_EXTI_LINE_PB1
  – LL_EXTI_LINE_PB2
  – LL_EXTI_LINE_PB3
  – LL_EXTI_LINE_PB4
  – LL_EXTI_LINE_PB5
  – LL_EXTI_LINE_PB6
  – LL_EXTI_LINE_PB7
  – LL_EXTI_LINE_PB8
  – LL_EXTI_LINE_PB9
  – LL_EXTI_LINE_PB10
  – LL_EXTI_LINE_PB11
  – LL_EXTI_LINE_PB12
  – LL_EXTI_LINE_PB13
  – LL_EXTI_LINE_PB14
  – LL_EXTI_LINE_PB15
  – LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_DTR LL_EXTI_EnableEdgeDetection

LL_EXTI_DisableEdgeDetection

Function name

__STATIC_INLINE void LL_EXTI_DisableEdgeDetection (uint32_t ExtiLine)

Function description

Set IO Interrupt Type for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_PA0
  – LL_EXTI_LINE_PA1
  – LL_EXTI_LINE_PA2
  – LL_EXTI_LINE_PA3
  – LL_EXTI_LINE_PA4
  – LL_EXTI_LINE_PA5
  – LL_EXTI_LINE_PA6
  – LL_EXTI_LINE_PA7
  – LL_EXTI_LINE_PA8
  – LL_EXTI_LINE_PA9
  – LL_EXTI_LINE_PA10
  – LL_EXTI_LINE_PA11
  – LL_EXTI_LINE_PA12
  – LL_EXTI_LINE_PA13
  – LL_EXTI_LINE_PA14
  – LL_EXTI_LINE_PA15
  – LL_EXTI_LINE_PB0
  – LL_EXTI_LINE_PB1
  – LL_EXTI_LINE_PB2
  – LL_EXTI_LINE_PB3
  – LL_EXTI_LINE_PB4
  – LL_EXTI_LINE_PB5
  – LL_EXTI_LINE_PB6
  – LL_EXTI_LINE_PB7
  – LL_EXTI_LINE_PB8
  – LL_EXTI_LINE_PB9
  – LL_EXTI_LINE_PB10
  – LL_EXTI_LINE_PB11
  – LL_EXTI_LINE_PB12
  – LL_EXTI_LINE_PB13
  – LL_EXTI_LINE_PB14
  – LL_EXTI_LINE_PB15
  – LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_DTR LL_EXTI_DisableEdgeDetection

LL_EXTI_IsEnabledEdgeDetection

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledEdgeDetection (uint32_t ExtiLine)

Function description

Get IO Interrupt Type for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be one of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL al One of the following values:
  - LL_EXTI_TYPE_EDGE
  - LL_EXTI_TYPE_LEVEL

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_DTR LL_EXTI_IsEnabledEdgeDetection

LL_EXTI_EnableBothEdgeTrig

Function name

__STATIC_INLINE void LL_EXTI_EnableBothEdgeTrig (uint32_t ExtiLine)

Function description

Enable ExtiLine Rising Edge Trigger for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_PA0
  – LL_EXTI_LINE_PA1
  – LL_EXTI_LINE_PA2
  – LL_EXTI_LINE_PA3
  – LL_EXTI_LINE_PA4
  – LL_EXTI_LINE_PA5
  – LL_EXTI_LINE_PA6
  – LL_EXTI_LINE_PA7
  – LL_EXTI_LINE_PA8
  – LL_EXTI_LINE_PA9
  – LL_EXTI_LINE_PA10
  – LL_EXTI_LINE_PA11
  – LL_EXTI_LINE_PA12
  – LL_EXTI_LINE_PA13
  – LL_EXTI_LINE_PA14
  – LL_EXTI_LINE_PA15
  – LL_EXTI_LINE_PB0
  – LL_EXTI_LINE_PB1
  – LL_EXTI_LINE_PB2
  – LL_EXTI_LINE_PB3
  – LL_EXTI_LINE_PB4
  – LL_EXTI_LINE_PB5
  – LL_EXTI_LINE_PB6
  – LL_EXTI_LINE_PB7
  – LL_EXTI_LINE_PB8
  – LL_EXTI_LINE_PB9
  – LL_EXTI_LINE_PB10
  – LL_EXTI_LINE_PB11
  – LL_EXTI_LINE_PB12
  – LL_EXTI_LINE_PB13
  – LL_EXTI_LINE_PB14
  – LL_EXTI_LINE_PB15
  – LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_IBER LL_EXTI_EnableBothEdgeTrig

LL_EXTI_DisableBothEdgeTrig

Function name

__STATIC_INLINE void LL_EXTI_DisableBothEdgeTrig (uint32_t ExtiLine)

Function description

Disable ExtiLine Rising Edge Trigger for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IBER LL_EXTI_DisableBothEdgeTrig

LL_EXTI_IsEnabledBothEdgeTrig

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledBothEdgeTrig (uint32_t ExtiLine)

Function description

Check if rising edge trigger is enabled for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

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

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IBER LL_EXTI_IsEnabledBothEdgeTrig

LL_EXTI_EnableRisingTrig

Function name

__STATIC_INLINE void LL_EXTI_EnableRisingTrig (uint32_t ExtiLine)

Function description

Enable ExtiLine Falling Edge Trigger for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_PA0
  – LL_EXTI_LINE_PA1
  – LL_EXTI_LINE_PA2
  – LL_EXTI_LINE_PA3
  – LL_EXTI_LINE_PA4
  – LL_EXTI_LINE_PA5
  – LL_EXTI_LINE_PA6
  – LL_EXTI_LINE_PA7
  – LL_EXTI_LINE_PA8
  – LL_EXTI_LINE_PA9
  – LL_EXTI_LINE_PA10
  – LL_EXTI_LINE_PA11
  – LL_EXTI_LINE_PA12
  – LL_EXTI_LINE_PA13
  – LL_EXTI_LINE_PA14
  – LL_EXTI_LINE_PA15
  – LL_EXTI_LINE_PB0
  – LL_EXTI_LINE_PB1
  – LL_EXTI_LINE_PB2
  – LL_EXTI_LINE_PB3
  – LL_EXTI_LINE_PB4
  – LL_EXTI_LINE_PB5
  – LL_EXTI_LINE_PB6
  – LL_EXTI_LINE_PB7
  – LL_EXTI_LINE_PB8
  – LL_EXTI_LINE_PB9
  – LL_EXTI_LINE_PB10
  – LL_EXTI_LINE_PB11
  – LL_EXTI_LINE_PB12
  – LL_EXTI_LINE_PB13
  – LL_EXTI_LINE_PB14
  – LL_EXTI_LINE_PB15
  – LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IO_IEVR LL_EXTI_EnableRisingTrig

LL_EXTI_DisableRisingTrig

Function name

__STATIC_INLINE void LL_EXTI_DisableRisingTrig (uint32_t ExtiLine)

Function description

Disable ExtiLine Falling Edge Trigger for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_IEVR LL_EXTI_DisableRisingTrig

LL_EXTI_IsEnabledRisingTrig

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsEnabledRisingTrig (uint32_t ExtiLine)

Function description

Check if falling edge trigger is enabled for Lines in range PA0...

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

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

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09
- Please check each device line mapping for EXTI Line availability

Reference Manual to LL API cross reference:

- IO_IEVR LL_EXTI_IsEnabledRisingTrig

LL_EXTI_IsActiveFlag

Function name

__STATIC_INLINE uint32_t LL_EXTI_IsActiveFlag (uint32_t ExtiLine)

Function description

Check if the ExtLine Flag is set or not for Lines in range PA0...PB15.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

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

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_ISCR LL_EXTI_IsActiveFlag

LL_EXTI_ReadFlag

Function name

__STATIC_INLINE uint32_t LL_EXTI_ReadFlag (uint32_t ExtiLine)

Function description

Read ExtLine Combination Flag for Lines in range PA0...PB15.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  – LL_EXTI_LINE_PA0
  – LL_EXTI_LINE_PA1
  – LL_EXTI_LINE_PA2
  – LL_EXTI_LINE_PA3
  – LL_EXTI_LINE_PA4
  – LL_EXTI_LINE_PA5
  – LL_EXTI_LINE_PA6
  – LL_EXTI_LINE_PA7
  – LL_EXTI_LINE_PA8
  – LL_EXTI_LINE_PA9
  – LL_EXTI_LINE_PA10
  – LL_EXTI_LINE_PA11
  – LL_EXTI_LINE_PA12
  – LL_EXTI_LINE_PA13
  – LL_EXTI_LINE_PA14
  – LL_EXTI_LINE_PA15
  – LL_EXTI_LINE_PB0
  – LL_EXTI_LINE_PB1
  – LL_EXTI_LINE_PB2
  – LL_EXTI_LINE_PB3
  – LL_EXTI_LINE_PB4
  – LL_EXTI_LINE_PB5
  – LL_EXTI_LINE_PB6
  – LL_EXTI_LINE_PB7
  – LL_EXTI_LINE_PB8
  – LL_EXTI_LINE_PB9
  – LL_EXTI_LINE_PB10
  – LL_EXTI_LINE_PB11
  – LL_EXTI_LINE_PB12
  – LL_EXTI_LINE_PB13
  – LL_EXTI_LINE_PB14
  – LL_EXTI_LINE_PB15
  – LL_EXTI_LINE_ALL

Return values

- **Flag:** state

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_ISCR LL_EXTI_ReadFlag

LL_EXTI_ClearFlag

Function name

__STATIC_INLINE void LL_EXTI_ClearFlag (uint32_t ExtiLine)

Function description

Clear ExtLine Flags for Lines in range PA0...PB15.

Parameters

- **ExtiLine:** This parameter can be a combination of the following values:
  - LL_EXTI_LINE_PA0
  - LL_EXTI_LINE_PA1
  - LL_EXTI_LINE_PA2
  - LL_EXTI_LINE_PA3
  - LL_EXTI_LINE_PA4
  - LL_EXTI_LINE_PA5
  - LL_EXTI_LINE_PA6
  - LL_EXTI_LINE_PA7
  - LL_EXTI_LINE_PA8
  - LL_EXTI_LINE_PA9
  - LL_EXTI_LINE_PA10
  - LL_EXTI_LINE_PA11
  - LL_EXTI_LINE_PA12
  - LL_EXTI_LINE_PA13
  - LL_EXTI_LINE_PA14
  - LL_EXTI_LINE_PA15
  - LL_EXTI_LINE_PB0
  - LL_EXTI_LINE_PB1
  - LL_EXTI_LINE_PB2
  - LL_EXTI_LINE_PB3
  - LL_EXTI_LINE_PB4
  - LL_EXTI_LINE_PB5
  - LL_EXTI_LINE_PB6
  - LL_EXTI_LINE_PB7
  - LL_EXTI_LINE_PB8
  - LL_EXTI_LINE_PB9
  - LL_EXTI_LINE_PB10
  - LL_EXTI_LINE_PB11
  - LL_EXTI_LINE_PB12
  - LL_EXTI_LINE_PB13
  - LL_EXTI_LINE_PB14
  - LL_EXTI_LINE_PB15
  - LL_EXTI_LINE_ALL

Return values

- **None:**

Notes

- PA0-PA15 and PB0-PB15 are valid for STM32WB06 and STM32WB07
- PA0-PA3, PA8-PA11, PB0-PB7 and PB12-PB15 are valid for STM32WB05 and STM32WB09

Reference Manual to LL API cross reference:

- IO_ISCR LL_EXTI_ClearFlag

LL_FLASH_EnableBurstWrite

Function name

__STATIC_INLINE void LL_FLASH_EnableBurstWrite (FLASH_TypeDef * FLASHx)

Function description

Enable the FLASH burst write operations.

Parameters

- **FLASHx:** FLASH Instance

Return values

- **None:**

LL_FLASH_DisableBurstWrite

Function name

__STATIC_INLINE void LL_FLASH_DisableBurstWrite (FLASH_TypeDef * FLASHx)

Function description

Disable the FLASH burst write operations.

Parameters

- **FLASHx:** FLASH Instance

Return values

- **None:**

LL_FLASH_SetVectTable

Function name

__STATIC_INLINE void LL_FLASH_SetVectTable (FLASH_TypeDef * FLASHx, uint32_t vect_table)

Function description

Set the FLASH Interrupt Vector Table location.

Parameters

- **FLASHx:** FLASH Instance
- **vect_table:** Vector Table Configuration This parameter can be one of the following values :
  – LL_SYSCFG_REMAP_FLASH Vector Table located in FLASH
  – LL_SYSCFG_REMAP_SRAM Vector Table located in RAM

Return values

- **None:**

LL_FLASH_GetVectTable

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetVectTable (FLASH_TypeDef * FLASHx)

Function description

Get the FLASH Interrupt Vector Table location.

Parameters

- **FLASHx:** FLASH Instance

Return values

- **Vector:** Table Configuration Returned value can be one of the following values :
  – LL_SYSCFG_REMAP_FLASH Vector Table located in FLASH
  – LL_SYSCFG_REMAP_SRAM Vector Table located in RAM

LL_FLASH_EnableIT

Function name

__STATIC_INLINE void LL_FLASH_EnableIT (FLASH_TypeDef * FLASHx, uint32_t interrupt)

Function description

Enable the specified FLASH interrupt.

Parameters

- **FLASHx:** FLASH Instance
- **interrupt:** FLASH interrupt This parameter can be any combination of the following values:
  – LL_FLASH_IT_CMDDONE Command Done Interrupt
  – LL_FLASH_IT_CMDSTART Command Started Interrupt
  – LL_FLASH_IT_CMDERR Command Error Interrupt
  – LL_FLASH_IT_ILLCMD Illegal Command Interrupt

Return values

- **none:**

LL_FLASH_DisableIT

Function name

__STATIC_INLINE void LL_FLASH_DisableIT (FLASH_TypeDef * FLASHx, uint32_t interrupt)

Function description

Disable the specified FLASH interrupt.

Parameters

- **FLASHx:** FLASH Instance
- **interrupt:** FLASH interrupt This parameter can be any combination of the following values:
  – LL_FLASH_IT_CMDDONE Command Done Interrupt
  – LL_FLASH_IT_CMDSTART Command Started Interrupt
  – LL_FLASH_IT_CMDERR Command Error Interrupt
  – LL-FLASH_IT_ILLCMD Illegal Command Interrupt

Return values

- **none:**

LL_FLASH_GetIT

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetIT (FLASH_TypeDef * FLASHx, uint32_t interrupt)

Function description

Check whether the specified FLASH interrupt has occurred or not.

Parameters

- **FLASHx:** FLASH Instance
- **interrupt:** FLASH interrupt This parameter can be any combination of the following values:
  – FLASH_IT_CMDDONE Command Done Interrupt
  – FLASH_IT_CMDSTART Command Started Interrupt
  – FLASH_IT_CMDERR Command Error Interrupt
  – FLASH_IT_ILLCMD Illegal Command Interrupt

Return values

- **The:** new state of interrupt flag (SET or RESET).

LL_FLASH_ClearIT

Function name

__STATIC_INLINE void LL_FLASH_ClearIT (FLASH_TypeDef * FLASHx, uint32_t interrupt)

Function description

Clear the specified FLASH interrupt flag.

Parameters

- **FLASHx:** FLASH Instance
- **interrupt:** FLASH interrupt This parameter can be any combination of the following values:
  – FLASH_IT_CMDDONE Command Done Interrupt
  – FLASH_IT_CMDSTART Command Started Interrupt
  – FLASH_IT_CMDERR Command Error Interrupt
  – FLASH_IT_ILLCMD Illegal Command Interrupt

Return values

- **None:**

LL_FLASH_GetFlag

Function name

__STATIC_INLINE uint32_t LL_FLASH_GetFlag (FLASH_TypeDef * FLASHx, uint32_t flag)

Function description

Check whether the specified FLASH flag is set or not.

Parameters

- **FLASHx:** FLASH Instance
- **flag:** specifies the FLASH flag to check. This parameter can be one of the following values:
  – LL_FLASH_FLAG_CMDDONE FLASH command done flag
  – LL_FLASH_FLAG_CMDSTART FLASH command started flag
  – LL_FLASH_FLAG_CMDERR FLASH command error flag
  – LL_FLASH_FLAG_ILLCMD FLASH illegal command flag

Return values

- **The:** new state of FLASH flag (SET or RESET).

LL_FLASH_ClearFlag

Function name

__STATIC_INLINE void LL_FLASH_ClearFlag (FLASH_TypeDef * FLASHx, uint32_t flag)

Function description

Clear the FLASH's pending flags.

Parameters

- **FLASHx:** FLASH Instance
- **flag:** specifies the FLASH flags to clear. This parameter can be any combination of the following values:
  – LL_FLASH_FLAG_CMDDONE FLASH command done flag
  – LL_FLASH_FLAG_CMDSTART FLASH command started flag
  – LL_FLASH_FLAG_CMDERR FLASH command error flag
  – LL_FLASH_FLAG_ILLCMD FLASH illegal command flag

Return values

- **None:**

LL_FLASH_SWD_Access_IsDisabled

Function name

__STATIC_INLINE uint32_t LL_FLASH_SWD_Access_IsDisabled (void)

Function description

Check if FLASH & RAM SWD access is disabled.

Return values

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

Reference Manual to LL API cross reference:

- SIZE SWD_DISABLE LL_FLASH_SWD_Access_IsDisabled

LL_FLASH_ReadoutProtection_IsEnabled

Function name

__STATIC_INLINE uint32_t LL_FLASH_ReadoutProtection_IsEnabled (void)

Function description

Check if FLASH Readout Protection is enabled.

Return values

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

Reference Manual to LL API cross reference:

- SIZE FLASH_SECURE LL_FLASH_ReadoutProtection_IsEnabled

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_CONFIG WAIT_STATE 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_CONFIG WAIT_STATE LL_FLASH_GetLatency

LL_SYSCFG_BLERXTX_EnableIT

Function name

__STATIC_INLINE void LL_SYSCFG_BLERXTX_EnableIT (uint32_t Signal)

Function description

Enable Tx Rx Interrupt.

Parameters

- **Signal:** This parameter can be a combination of the following values:
  - LL_SYSCFG_BLE_TX_EVENT
  - LL_SYSCFG_BLE_RX_EVENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLERXTX_IER LL_SYSCFG_BLERXTX_EnableIT

LL_SYSCFG_BLERXTX_DisableIT

Function name

__STATIC_INLINE void LL_SYSCFG_BLERXTX_DisableIT (uint32_t Signal)

Function description

Disable Tx Rx Interrupt.

Parameters

- **Signal:** This parameter can be a combination of the following values:
  - LL_SYSCFG_BLE_TX_EVENT
  - LL_SYSCFG_BLE_RX_EVENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLERXTX_IER LL_SYSCFG_BLERXTX_DisableIT

LL_SYSCFG_BLERXTX_IsEnabledIT

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_BLERXTX_IsEnabledIT (uint32_t Signal)

Function description

Indicate if Tx Rx Interrupt is enabled.

Parameters

- **Signal:** This parameter can be a combination of the following values:
  – LL_SYSCFG_BLE_TX_EVENT
  – LL_SYSCFG_BLE_RX_EVENT

Return values

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

Reference Manual to LL API cross reference:

- BLERXTX_IER LL_SYSCFG_BLERXTX_IsEnabledIT

LL_SYSCFG_BLERXTX_SetType

Function name

__STATIC_INLINE void LL_SYSCFG_BLERXTX_SetType (uint8_t Type, uint32_t Signal)

Function description

Set Tx-Rx Interrupt Detection Type.

Parameters

- **Type:** This parameter can be one of the following values:
  – LL_SYSCFG_BLERXTX_DET_TYPE_EDGE
  – LL_SYSCFG_BLERXTX_DET_TYPE_LEVEL
- **Signal:** This parameter can be a combination of the following values:
  – LL_SYSCFG_BLE_TX_EVENT
  – LL_SYSCFG_BLE_RX_EVENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLERXTX_DTR LL_SYSCFG_BLERXTX_SetType

LL_SYSCFG_BLERXTX_GetType

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_BLERXTX_GetType (uint32_t Signal)

Function description

Get Tx-Rx Interrupt Detection Type.

Parameters

- **Signal:** This parameter one of the following values:
  – LL_SYSCFG_BLE_TX_EVENT
  – LL_SYSCFG_BLE_RX_EVENT

Return values

- **One:** of the following values:
  - LL_SYSCFG_BLERXTX_DET_TYPE_EDGE
  - LL_SYSCFG_BLERXTX_DET_TYPE_LEVEL

Reference Manual to LL API cross reference:

- BLERXTX_DTR LL_SYSCFG_BLERXTX_GetType

LL_SYSCFG_BLERXTX_SetTrigger

Function name

__STATIC_INLINE void LL_SYSCFG_BLERXTX_SetTrigger (uint8_t Trigger, uint32_t Signal)

Function description

Set Tx Rx Interrupt Trigger.

Parameters

- **Trigger:** This parameter can be one of the following values:
  - LL_SYSCFG_BLERXTX_TRIGGER_RISING_EDGE
  - LL_SYSCFG_BLERXTX_TRIGGER_FALLING_EDGE
  - LL_SYSCFG_BLERXTX_TRIGGER_BOTH_EDGE
  - LL_SYSCFG_BLERXTX_TRIGGER_LOW_LEVEL
  - LL_SYSCFG_BLERXTX_TRIGGER_HIGH_LEVEL
- **Signal:** This parameter can be a combination of the following values:
  - LL_SYSCFG_BLE_TX_EVENT
  - LL_SYSCFG_BLE_RX_EVENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLERXTX_IBER/BLERXTX__IEVR/BLERXTX__DTR LL_SYSCFG_BLERXTX_SetTrigger

LL_SYSCFG_BLERXTX_GetTrigger

Function name

__STATIC_INLINE uint8_t LL_SYSCFG_BLERXTX_GetTrigger (uint32_t Signal)

Function description

Get Tx Rx Interrupt Trigger.

Parameters

- **Signal:** This parameter can be one of the following values:
  - LL_SYSCFG_BLE_TX_EVENT
  - LL_SYSCFG_BLE_RX_EVENT

Return values

- **The:** return value can be one of the following values:
  - LL_SYSCFG_BLERXTX_TRIGGER_RISING_EDGE
  - LL_SYSCFG_BLERXTX_TRIGGER_FALLING_EDGE
  - LL_SYSCFG_BLERXTX_TRIGGER_BOTH_EDGE
  - LL_SYSCFG_BLERXTX_TRIGGER_LOW_LEVEL
  - LL_SYSCFG_BLERXTX_TRIGGER_HIGH_LEVEL

Reference Manual to LL API cross reference:

- BLERXTX_IBER/BLERXTX_IEVR/BLERXTX_DTR LL_SYSCFG_BLERXTX_GetTrigger

LL_SYSCFG_BLERXTX_IsInterruptPending

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_BLERXTX_IsInterruptPending (uint32_t Signal)

Function description

Check if the Tx Rx Interrupt is set or not.

Parameters

- **Signal:** This parameter can be a combination of the following values:
  - LL_SYSCFG_BLE_TX_EVENT
  - LL_SYSCFG_BLE_RX_EVENT

Return values

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

Notes

- This bit is cleared by writing a 1 to the bit.

Reference Manual to LL API cross reference:

- BLERXTX_ISCR LL_SYSCFG_BLERXTX_IsInterruptPending

LL_SYSCFG_BLERXTX_ClearInterrupt

Function name

__STATIC_INLINE void LL_SYSCFG_BLERXTX_ClearInterrupt (uint32_t Signal)

Function description

Clear the Tx Rx Interrupt.

Parameters

- **Signal:** This parameter can be a combination of the following values:
  - LL_SYSCFG_BLE_TX_EVENT
  - LL_SYSCFG_BLE_RX_EVENT

Return values

- **None:**

Reference Manual to LL API cross reference:

- BLERXTX_ISCR LL_SYSCFG_BLERXTX_ClearInterrupt

LL_SYSCFG_BLETX_EdgeSequence

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_BLETX_EdgeSequence (void)

Function description

Returns the TX interrupt edge status on a TX sequence event.

Return values

- **The:** return value can be one of the following values:
  - LL_SYSCFG_BLE_TX_FALLING_EDGE_EVENT
  - LL_SYSCFG_BLE_TX_RISING_EDGE_EVENT

Reference Manual to LL API cross reference:

- BLERXTX_ISCR LL_SYSCFG_BLETX_EdgeSequence

LL_SYSCFG_BLERX_EdgeSequence

Function name

__STATIC_INLINE uint32_t LL_SYSCFG_BLERX_EdgeSequence (void)

Function description

Returns the RX interrupt edge status on a RX sequence event.

Return values

- **The:** return value can be one of the following values:
  – LL_SYSCFG_BLE_RX_FALLING_EDGE_EVENT
  – LL_SYSCFG_BLE_RX_RISING_EDGE_EVENT

Reference Manual to LL API cross reference:

- BLERXTX_ISCR LL_SYSCFG_BLERX_EdgeSequence

LL_SYSCFG_IO_Init

Function name

ErrorStatus LL_SYSCFG_IO_Init (LL_SYSCFG_IO_InitTypeDef * SYSCFG_IO_InitStruct)

Function description

Initialize the SYSCFG_IO registers according to the specified parameters in SYSCFG_IO_InitStruct.

Parameters

- **SYSCFG_IO_InitStruct:** pointer to a LL_SYSCFG_IO_InitTypeDef structure.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: SYSCFG_IO registers are initialized
  – ERROR: not applicable

LL_SYSCFG_IO_DeInit

Function name

ErrorStatus LL_SYSCFG_IO_DeInit (void)

Function description

De-initialize the SYSCFG_IO registers to their default reset values.

Return values

- **An:** ErrorStatus enumeration value:
  – SUCCESS: SYSCFG_IO registers are de-initialized
  – ERROR: not applicable

LL_SYSCFG_IO_StructInit

Function name

void LL_SYSCFG_IO_StructInit (LL_SYSCFG_IO_InitTypeDef * SYSCFG_IO_InitStruct)

Function description

Set each LL_SYSCFG_IO_InitTypeDef field to default value.

Parameters

- **SYSCFG_IO_InitStruct:** Pointer to a LL_SYSCFG_IO_InitTypeDef structure.

Return values

- **None:**

62.3 SYSTEM Firmware driver defines

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

62.3.1 SYSTEM

SYSTEM

Device Version Information

LL_BLUENRG_LP_CUT_10

LL_BLUENRG_LP_CUT_20

FLASH Interrupts Definition

LL_FLASH_IT_CMDDONE

FLASH command done Interrupt source

LL_FLASH_IT_CMDSTART

FLASH command started Interrupt source

LL_FLASH_IT_CMDERR

FLASH command error Interrupt source

LL_FLASH_IT_ILLCMD

FLASH illegal command Interrupt source

FLASH Wait States

LL_FLASH_WAIT_STATES_0

FLASH Zero wait state

LL_FLASH_WAIT_STATES_1

FLASH One wait state

SYSCFG I2C FASTMODEPLUS

LL_SYSCFG_I2C_FASTMODEPLUS_PA0

Enable Fast Mode Plus on PA0

LL_SYSCFG_I2C_FASTMODEPLUS_PA1

Enable Fast Mode Plus on PA1

LL_SYSCFG_I2C_FASTMODEPLUS_PB6

Enable Fast Mode Plus on PB6

LL_SYSCFG_I2C_FASTMODEPLUS_PB7

Enable Fast Mode Plus on PB7

EXTI Lines

LL_EXTI_LINE_PA0
PA0

LL_EXTI_LINE_PA1
PA1

LL_EXTI_LINE_PA2
PA2

LL_EXTI_LINE_PA3
PA3

LL_EXTI_LINE_PA8
PA8

LL_EXTI_LINE_PA9
PA9

LL_EXTI_LINE_PA10
PA10

LL_EXTI_LINE_PA11
PA11

LL_EXTI_LINE_PB0
PB0

LL_EXTI_LINE_PB1
PB1

LL_EXTI_LINE_PB2
PB2

LL_EXTI_LINE_PB3
PB3

LL_EXTI_LINE_PB4
PB4

LL_EXTI_LINE_PB5
PB5

LL_EXTI_LINE_PB6
PB6

LL_EXTI_LINE_PB7
PB7

LL_EXTI_LINE_PB12
PB12

LL_EXTI_LINE_PB13
PB13

LL_EXTI_LINE_PB14
PB14

LL_EXTI_LINE_PB15

PB15

LL_EXTI_LINE_ALL

All the PA0...PB15

LL_EXTI_LINE_NONE

FLASH LATENCY

LL_FLASH_LATENCY_0

FLASH Zero wait state

LL_FLASH_LATENCY_1

FLASH One wait state when system clock frequency is 64 MHz

SYSCFG REMAP

LL_SYSCFG_REMAP_FLASH

Main flash memory mapped at 0x00000000

LL_SYSCFG_REMAP_SRAM

SRAM mapped at 0x00000000

EXTI Edge Trigger

LL_EXTI_TRIGGER_NONE

Trigger None

LL_EXTI_TRIGGER_RISING_EDGE

Trigger Rising Edge

LL_EXTI_TRIGGER_FALLING_EDGE

Trigger Falling Edge

LL_EXTI_TRIGGER_BOTH_EDGE

Trigger Both Edge

LL_EXTI_TRIGGER_LOW_LEVEL

Trigger Low Level

LL_EXTI_TRIGGER_HIGH_LEVEL

Trigger High Level

EXTI Type

LL_EXTI_TYPE_EDGE

Interrupt Type EDGE detection

LL_EXTI_TYPE_LEVEL

Interrupt Type LEVEL detection

SYSCFG Power Controller Interrupt MASK

LL_SYSCFG_PWRC_PVD

PVD interrupt mask

LL_SYSCFG_PWRC_WKUP

WKUP interrupt mask

63 LL TIM Generic Driver

63.1 TIM Firmware driver registers structures

63.1.1 LL_TIM_InitTypeDef

LL_TIM_InitTypeDef is defined in the stm32wb0x_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()**.

63.1.2 LL_TIM_OC_InitTypeDef

LL_TIM_OC_InitTypeDef is defined in the stm32wb0x_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()**.

63.1.3 LL_TIM_IC_InitTypeDef

LL_TIM_IC_InitTypeDef is defined in the stm32wb0x_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()**.

63.1.4 LL_TIM_ENCODER_InitTypeDef

LL_TIM_ENCODER_InitTypeDef is defined in the stm32wb0x_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()**.

63.1.5 LL_TIM_BDTR_InitTypeDef

LL_TIM_BDTR_InitTypeDef is defined in the stm32wb0x_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 BreakAFMode*
- *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::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::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.

63.2 TIM Firmware driver API description

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

63.2.1 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)

Return values

- **None:**

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)

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.
- 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

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_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

- 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

- 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

- 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

- 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_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

- 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

- 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

- 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

- 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_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

- 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

- 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

- 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

- 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_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_TI1F_ED
  – LL_TIM_TS_TI1FP1
  – LL_TIM_TS_TI2FP2
  – LL_TIM_TS_ETRF

Return values

- **None:**

Notes

- Macro IS_TIM_SLAVE_INSTANCE(TIMx) can be used to check whether or not a timer instance can operate as a slave timer.

Reference Manual to LL API cross reference:

- SMCR TS LL_TIM_SetTriggerInput

LL_TIM_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_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 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.
- 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.

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_ReArmBRK

Function name

__STATIC_INLINE void LL_TIM_ReArmBRK (TIM_TypeDef * TIMx)

Function description

Re-arm the break input (when it operates in bidirectional mode).

Parameters

- **TIMx:** Timer instance

Return values

- **None:**

Notes

- The Break input is automatically armed as soon as MOE bit is set.

Reference Manual to LL API cross reference:

- BDTR BKDSRM LL_TIM_ReArmBRK

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_OR1
  - 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
- **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:** This parameter can be one of the following values:
  – LL_TIM_TIM17_TI1_RMP_GPIO
  – LL_TIM_TIM17_TI1_RMP_LCO
  – LL_TIM_TIM17_TI1_RMP_MCO

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:

- TIM17_OR1 TI1_RMP 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_OCREF_CLR
  – 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_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_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_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_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_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_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.

63.3 TIM Firmware driver defines

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

63.3.1 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

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 INPUT

LL_TIM_BREAK_INPUT_BKIN

TIMx_BKIN 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_CH3

Timer input/output channel 3

LL_TIM_CHANNEL_CH4

Timer input/output channel 4

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_OR1
TIMx_OR1 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

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 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_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

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_OCREF_CLR

OCREF_CLR_INT is connected to the OCREF_CLR input

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

TIM17 External Input Ch1 Remap

LL_TIM_TIM17_TI1_RMP_GPIO

TIM17_TI1 is connected to GPIO

LL_TIM_TIM17_TI1_RMP_LCO

TIM17_TI1 is connected to LCO

LL_TIM_TIM17_TI1_RMP_MCO

TIM17_TI1 is connected to MCO

Trigger Selection

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 ());

64 LL USART Generic Driver

64.1 USART Firmware driver registers structures

64.1.1 LL_USART_InitTypeDef

LL_USART_InitTypeDef is defined in the stm32wb0x_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()**.

64.1.2 LL_USART_ClockInitTypeDef

LL_USART_ClockInitTypeDef is defined in the stm32wb0x_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.

64.2 USART Firmware driver API description

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

64.2.1 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:**

64.3 USART Firmware driver defines

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

64.3.1 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

65 LL UTILS Generic Driver

65.1 UTILS Firmware driver registers structures

65.1.1 LL_UTILS_ClkInitTypeDef

LL_UTILS_ClkInitTypeDef is defined in the stm32wb0x_ll_utils.h

Data Fields

- ***uint32_t dummy***

Field Documentation

- ***uint32_t LL_UTILS_ClkInitTypeDef::dummy***

65.2 UTILS Firmware driver API description

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

65.2.1 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 64 bits)

Return values

- **Word0:** of UID64

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 64 bits)

Return values

- **Word1:** of UID64

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 words number of the device flash memory expressed. As an example, 0x7FFF corresponds to 128 Kbytes.

LL_GetRAMSize

Function name

__STATIC_INLINE uint32_t LL_GetRAMSize (void)

Function description

Get RAM memory size.

Return values

- **Returned:** value can be one of the following values:
  - LL_UTILS_RAMSIZE_24K
  - LL_UTILS_RAMSIZE_32K
  - LL_UTILS_RAMSIZE_48K
  - LL_UTILS_RAMSIZE_64K

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_QFN48
  - LL_UTILS_PACKAGETYPE_QFN32
  - LL_UTILS_PACKAGETYPE_CSP49

LL_GetDIE_ID

Function name

__STATIC_INLINE uint32_t LL_GetDIE_ID (void)

Function description

Get Device version and cut information.

Return values

- **DIE_ID:** register information

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:** SysClk frequency in Hz
- **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:** SysClk frequency in Hz

Return values

- **None:**

Notes

- When a RTOS is used, it is recommended to avoid changing the Systick configuration by calling this function, for a delay use rather osDelay RTOS service.

LL_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 the System Core Clock.

Parameters

- **HCLKFrequency:** system clock divided factor from HSI64MPLL This parameter can be one of the following values: SYSCLK_64M SYSCLK_32M SYSCLK_16M SYSCLK_8M SYSCLK_4M SYSCLK_2M SYSCLK_1M SYSCLK_DIRECT_HSE SYSCLK_DIRECT_HSE_DIV2

Return values

- **SUCCESS:** or error code

LL_GetSystemCoreClock

Function name

uint32_t LL_GetSystemCoreClock (void)

Function description

This function return the System Core Clock expressed in Hz.

Return values

- **System:** Core Clock frequency.

65.3 UTILS Firmware driver defines

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

65.3.1 UTILS

UTILS

HSE Bypass activation

LL_UTILS_HSEBYPASS_OFF

HSE Bypass is not enabled

LL_UTILS_HSEBYPASS_ON

HSE Bypass is enabled

PACKAGE TYPE

LL_UTILS_PACKAGETYPE_QFN48

QFN48 package type

LL_UTILS_PACKAGETYPE_QFN32

QFN32 package type

LL_UTILS_PACKAGETYPE_CSP49

CSP49 package type

RAM SIZE

LL_UTILS_RAMSIZE_24K

24 kB of RAM available

LL_UTILS_RAMSIZE_32K

32 kB of RAM available

LL_UTILS_RAMSIZE_48K

48 kB of RAM available

LL_UTILS_RAMSIZE_64K

64 kB of RAM available

66 Correspondence between API registers and API low-layer driver functions

This appendix contains the correspondence table between the register or bit name, as mentioned inside the reference manual of the component, and the name of the LL functions to modify or read this register or bit.

66.1 ADC

Table 25. Correspondence between ADC registers and ADC low-layer driver functions

Register	Field	Function
COMP_1	GAIN1	Section 44.2.1: LL_ADC_GetCalibPoint1Gain
		LL_ADC_SetCalibPointGain
	OFFSET1	Section 44.2.1: LL_ADC_GetCalibPoint1Offset
		Section 44.2.1: LL_ADC_SetCalibPoint1Offset
COMP_SEL	ADC_COMP_SEL_OFFSET_GAIN6	LL_ADC_GetCalibPointForSingleNeg
		LL_ADC_SetCalibPointForSingleNeg
	ADC_COMP_SEL_OFFSET_GAIN7	Section 44.2.1: LL_ADC_GetCalibPointForSinglePos3V6
		Section 44.2.1: LL_ADC_SetCalibPointForSinglePos3V6
	ADC_COMP_SEL_OFFSET_GAIN8	LL_ADC_GetCalibPointForDiff
		LL_ADC_SetCalibPointForDiff
CONF	ADC_CONT_1V2	LL_ADC_GetInputSamplingMode
		LL_ADC_SetInputSamplingMode
	BIT_INVERT_DIFF	LL_ADC_InvertOutputDiffModeDisable
		LL_ADC_InvertOutputDiffModeEnable
		LL_ADC_IsInvertOutputDiffModeEnabled
	BIT_INVERT_SN	LL_ADC_InvertOutputSingleNegModeDisable
		LL_ADC_InvertOutputSingleNegModeEnable
		LL_ADC_IsInvertOutputSingleNegModeEnabled
	CONT	LL_ADC_ContinuousModeDisable
		LL_ADC_ContinuousModeEnable
		LL_ADC_IsContinuousModeEnabled
	DMA_DS_ENA	LL_ADC_DMAModeDSDisable
		LL_ADC_DMAModeDSEnable
		LL_ADC_IsDMAModeDSEnabled
	OVR_DS_CFG	LL_ADC_GetOverrunDS
		LL_ADC_SetOverrunDS
	SAMPLE_RATE	LL_ADC_GetSampleRate
		LL_ADC_SetSampleRate
	SEQUENCE	LL_ADC_IsSequenceModeEnabled
		LL_ADC_SequenceModeDisable
		LL_ADC_SequenceModeEnable
	SEQ_LEN	LL_ADC_GetSequenceLength
		LL_ADC_SetSequenceLength

Register	Field	Function
CONF	SMPS_SYNCHRO_ENA	LL_ADC_IsSMPSSyncEnabled
		LL_ADC_SMPSSyncDisable
		LL_ADC_SMPSSyncEnable
CTRL	ADC_LDO_ENA	LL_ADC_DisableInternalRegulator
		LL_ADC_EnableInternalRegulator
		LL_ADC_IsInternalRegulatorEnabled
	ADC_ON_OFF	LL_ADC_Disable
		LL_ADC_Enable
		LL_ADC_IsConversionOngoing
		LL_ADC_IsEnabled
	START_CONV	LL_ADC_StartConversion
	STOP_OP_MODE	LL_ADC_IsStopConversionOngoing
		LL_ADC_StopConversion
DS_CONF	DS_RATIO	LL_ADC_GetDSDataOutputRatio
		LL_ADC_SetDSDataOutputRatio
	DS_WIDTH	LL_ADC_GetDSDataOutputWidth
		LL_ADC_SetDSDataOutputWidth
DS_DATAOUT	DS_DATA	LL_ADC_DSGetOutputData
IRQ_ENABLE	AWD_IRQ_ENA	LL_ADC_DisableIT_AWD
		LL_ADC_EnableIT_AWD
		LL_ADC_IsEnabledIT_AWD
	EOC_IRQ_ENA	LL_ADC_DisableIT_EOC
		LL_ADC_EnableIT_EOC
		LL_ADC_IsEnabledIT_EOC
	EODS_IRQ_ENA	LL_ADC_DisableIT_EODS
		LL_ADC_EnableIT_EODS
		LL_ADC_IsEnabledIT_EODS
	EOS_IRQ_ENA	LL_ADC_DisableIT_EOS
		LL_ADC_EnableIT_EOS
		LL_ADC_IsEnabledIT_EOS
	OVR_DS_IRQ_ENA	LL_ADC_DisableIT_DSOVR
		LL_ADC_EnableIT_DSOVR
		LL_ADC_IsEnabledIT_DSOVR
IRQ_STATUS	AWD_IRQ	LL_ADC_ClearFlag_AWD
		LL_ADC_IsActiveFlag_AWD
	EOC_IRQ	LL_ADC_ClearFlag_EOC
		LL_ADC_IsActiveFlag_EOC
	EODS_IRQ	LL_ADC_ClearFlag_EODS
		LL_ADC_IsActiveFlag_EODS
	EOS_IRQ	LL_ADC_ClearFlag_EOS
		LL_ADC_IsActiveFlag_EOS

Register	Field	Function
IRQ_STATUS	OVR_DS_IRQ	LL_ADC_ClearFlag_OVRDS
		LL_ADC_IsActiveFlag_OVRDS
SEQ_1	SEQ0	LL_ADC_GetSequencerRanks
	SEQ1	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ2	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ3	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ4	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ5	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ6	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ7	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
SEQ_2	SEQ10	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ11	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ12	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ13	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ14	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ15	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ8	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
	SEQ9	LL_ADC_GetSequencerRanks
		LL_ADC_SetSequencerRanks
SWITCH	SE_VIN_0	LL_ADC_SetChannelVoltageRange
	SE_VIN_1	LL_ADC_SetChannelVoltageRange
	SE_VIN_2	LL_ADC_GetChannelVoltageRange
		LL_ADC_SetChannelVoltageRange
	SE_VIN_3	LL_ADC_GetChannelVoltageRange
		LL_ADC_SetChannelVoltageRange
	SE_VIN_4	LL_ADC_GetChannelVoltageRange
		LL_ADC_SetChannelVoltageRange

Register	Field	Function
SWITCH	SE_VIN_5	LL_ADC_GetChannelVoltageRange
		LL_ADC_SetChannelVoltageRange
	SE_VIN_6	LL_ADC_GetChannelVoltageRange
		LL_ADC_SetChannelVoltageRange
	SE_VIN_7	LL_ADC_GetChannelVoltageRange
		LL_ADC_SetChannelVoltageRange
TIMER_CONF	ADC_LDO_DELAY	LL_ADC_GetADCLDODelay
		LL_ADC_SetADCLDODelay
WD_CONF	AWD_CHX	LL_ADC_GetAWDInputChannels
		LL_ADC_SetAWDInputChannels
WD_TH	ADC_WD_TH_WD_HT	LL_ADC_GetAWDThresholdHigh
		LL_ADC_SetAWDThresholdHigh
	ADC_WD_TH_WD_LT	LL_ADC_GetAWDThresholdLow
		LL_ADC_SetAWDThresholdLow

66.2 BUS

Table 26. Correspondence between BUS registers and BUS low-layer driver functions

Register	Field	Function
APB2ENR	MRBLEEN	LL_APB2_GRP1_DisableClock
		LL_APB2_GRP1_EnableClock
		LL_APB2_GRP1_IsEnabledClock
APB2RSTR	MRBLERST	LL_APB2_GRP1_ForceReset
		LL_APB2_GRP1_ReleaseReset

66.3 CORTEX

Table 27. Correspondence between CORTEX registers and CORTEX low-layer driver functions

Register	Field	Function
MPU_CTRL	ENABLE	LL_MPU_Disable
		LL_MPU_Enable
		LL_MPU_IsEnabled
MPU_RASR	AP	LL_MPU_ConfigRegion
	B	LL_MPU_ConfigRegion
	C	LL_MPU_ConfigRegion
	ENABLE	LL_MPU_DisableRegion
		LL_MPU_EnableRegion
	S	LL_MPU_ConfigRegion
	SIZE	LL_MPU_ConfigRegion
	XN	LL_MPU_ConfigRegion
MPU_RBAR	ADDR	LL_MPU_ConfigRegion

Register	Field	Function
MPU_RBAR	REGION	LL_MPU_ConfigRegion
MPU_RNR	REGION	LL_MPU_ConfigRegion
		LL_MPU_DisableRegion
SCB_CPUID	ARCHITECTURE	LL_CPUID_GetConstant
	IMPLEMENTER	LL_CPUID_GetImplementer
	PARTNO	LL_CPUID_GetParNo
	REVISION	LL_CPUID_GetRevision
	VARIANT	LL_CPUID_GetVariant
SCB_SCR	SLEEPDEEP	LL_LPM_EnableDeepSleep
		LL_LPM_EnableSleep
STK_CTRL	CLKSOURCE	LL_SYSTICK_GetClkSource
		LL_SYSTICK_SetClkSource
	COUNTFLAG	LL_SYSTICK_IsActiveCounterFlag
	DISABLE	LL_SYSTICK_Disable
	ENABLE	LL_SYSTICK_Enable
	TICKINT	LL_SYSTICK_DisableIT
		LL_SYSTICK_EnableIT
		LL_SYSTICK_IsEnabledIT

66.4 CRC

Table 28. **Correspondence between CRC registers and CRC low-layer driver functions**

Register	Field	Function
CR	POLYSIZE	LL_CRC_GetPolynomialSize
		LL_CRC_SetPolynomialSize
	RESET	LL_CRC_ResetCRCCalculationUnit
	REV_IN	LL_CRC_GetInputDataReverseMode
		LL_CRC_SetInputDataReverseMode
	REV_OUT	LL_CRC_GetOutputDataReverseMode
		LL_CRC_SetOutputDataReverseMode
DR	DR	LL_CRC_FeedData16
		LL_CRC_FeedData32
		LL_CRC_FeedData8
		LL_CRC_ReadData16
		LL_CRC_ReadData32
		LL_CRC_ReadData7
		LL_CRC_ReadData8
IDR	IDR	LL_CRC_Read_IDR
		LL_CRC_Write_IDR
INIT	INIT	LL_CRC_GetInitialData
		LL_CRC_SetInitialData

Register	Field	Function
POL	POL	LL_CRC_GetPolynomialCoef
		LL_CRC_SetPolynomialCoef

66.5 DMA

Table 29. Correspondence between DMA registers and DMA low-layer driver functions

Register	Field	Function
CCR	CIRC	LL_DMA_ConfigTransfer
		LL_DMA_GetMode
		LL_DMA_SetMode
	DIR	LL_DMA_ConfigTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	EN	LL_DMA_DisableChannel
		LL_DMA_EnableChannel
		LL_DMA_IsEnabledChannel
	HTIE	LL_DMA_DisableIT_HT
		LL_DMA_EnableIT_HT
		LL_DMA_IsEnabledIT_HT
	MEM2MEM	LL_DMA_ConfigTransfer
		LL_DMA_GetDataTransferDirection
		LL_DMA_SetDataTransferDirection
	MINC	LL_DMA_ConfigTransfer
		LL_DMA_GetMemoryIncMode
		LL_DMA_SetMemoryIncMode
	MSIZE	LL_DMA_ConfigTransfer
		LL_DMA_GetMemorySize
		LL_DMA_SetMemorySize
	PINC	LL_DMA_ConfigTransfer
		LL_DMA_GetPeriphIncMode
		LL_DMA_SetPeriphIncMode
	PL	LL_DMA_ConfigTransfer
		LL_DMA_GetChannelPriorityLevel
		LL_DMA_SetChannelPriorityLevel
	PSIZE	LL_DMA_ConfigTransfer
		LL_DMA_GetPeriphSize
		LL_DMA_SetPeriphSize
	TCIE	LL_DMA_DisableIT_TC
		LL_DMA_EnableIT_TC
		LL_DMA_IsEnabledIT_TC
	TEIE	LL_DMA_DisableIT_TE

Register	Field	Function
CCR	TEIE	LL_DMA_EnableIT_TE
		LL_DMA_IsEnabledIT_TE
CMAR	MA	LL_DMA_ConfigAddresses
		LL_DMA_GetM2MDstAddress
		LL_DMA_GetMemoryAddress
		LL_DMA_SetM2MDstAddress
		LL_DMA_SetMemoryAddress
CNDTR	NDT	LL_DMA_GetDataLength
		LL_DMA_SetDataLength
CPAR	PA	LL_DMA_ConfigAddresses
		LL_DMA_GetM2MSrcAddress
		LL_DMA_GetPeriphAddress
		LL_DMA_SetM2MSrcAddress
		LL_DMA_SetPeriphAddress
CxCR	DMAREQ_ID	LL_DMA_GetPeriphRequest
		LL_DMA_SetPeriphRequest
IFCR	CGIF1	LL_DMA_ClearFlag_GI1
	CGIF2	LL_DMA_ClearFlag_GI2
	CGIF3	LL_DMA_ClearFlag_GI3
	CGIF4	LL_DMA_ClearFlag_GI4
	CGIF5	LL_DMA_ClearFlag_GI5
	CGIF6	LL_DMA_ClearFlag_GI6
	CGIF7	LL_DMA_ClearFlag_GI7
	CGIF8	LL_DMA_ClearFlag_GI8
	CHTIF1	LL_DMA_ClearFlag_HT1
	CHTIF2	LL_DMA_ClearFlag_HT2
	CHTIF3	LL_DMA_ClearFlag_HT3
	CHTIF4	LL_DMA_ClearFlag_HT4
	CHTIF5	LL_DMA_ClearFlag_HT5
	CHTIF6	LL_DMA_ClearFlag_HT6
	CHTIF7	LL_DMA_ClearFlag_HT7
	CHTIF8	LL_DMA_ClearFlag_HT8
	CTCIF1	LL_DMA_ClearFlag_TC1
	CTCIF2	LL_DMA_ClearFlag_TC2
	CTCIF3	LL_DMA_ClearFlag_TC3
	CTCIF4	LL_DMA_ClearFlag_TC4
	CTCIF5	LL_DMA_ClearFlag_TC5
	CTCIF6	LL_DMA_ClearFlag_TC6
	CTCIF7	LL_DMA_ClearFlag_TC7
	CTCIF8	LL_DMA_ClearFlag_TC8
	CTEIF1	LL_DMA_ClearFlag_TE1

Register	Field	Function
IFCR	CTEIF2	LL_DMA_ClearFlag_TE2
	CTEIF3	LL_DMA_ClearFlag_TE3
	CTEIF4	LL_DMA_ClearFlag_TE4
	CTEIF5	LL_DMA_ClearFlag_TE5
	CTEIF6	LL_DMA_ClearFlag_TE6
	CTEIF7	LL_DMA_ClearFlag_TE7
	CTEIF8	LL_DMA_ClearFlag_TE8
ISR	GIF1	LL_DMA_IsActiveFlag_GI1
	GIF2	LL_DMA_IsActiveFlag_GI2
	GIF3	LL_DMA_IsActiveFlag_GI3
	GIF4	LL_DMA_IsActiveFlag_GI4
	GIF5	LL_DMA_IsActiveFlag_GI5
	GIF6	LL_DMA_IsActiveFlag_GI6
	GIF7	LL_DMA_IsActiveFlag_GI7
	GIF8	LL_DMA_IsActiveFlag_GI8
	HTIF1	LL_DMA_IsActiveFlag_HT1
	HTIF2	LL_DMA_IsActiveFlag_HT2
	HTIF3	LL_DMA_IsActiveFlag_HT3
	HTIF4	LL_DMA_IsActiveFlag_HT4
	HTIF5	LL_DMA_IsActiveFlag_HT5
	HTIF6	LL_DMA_IsActiveFlag_HT6
	HTIF7	LL_DMA_IsActiveFlag_HT7
	HTIF8	LL_DMA_IsActiveFlag_HT8
	TCIF1	LL_DMA_IsActiveFlag_TC1
	TCIF2	LL_DMA_IsActiveFlag_TC2
	TCIF3	LL_DMA_IsActiveFlag_TC3
	TCIF4	LL_DMA_IsActiveFlag_TC4
	TCIF5	LL_DMA_IsActiveFlag_TC5
	TCIF6	LL_DMA_IsActiveFlag_TC6
	TCIF7	LL_DMA_IsActiveFlag_TC7
	TCIF8	LL_DMA_IsActiveFlag_TC8
	TEIF1	LL_DMA_IsActiveFlag_TE1
	TEIF2	LL_DMA_IsActiveFlag_TE2
	TEIF3	LL_DMA_IsActiveFlag_TE3
	TEIF4	LL_DMA_IsActiveFlag_TE4
	TEIF5	LL_DMA_IsActiveFlag_TE5
	TEIF6	LL_DMA_IsActiveFlag_TE6
	TEIF7	LL_DMA_IsActiveFlag_TE7
	TEIF8	LL_DMA_IsActiveFlag_TE8

66.6 DMAMUX

Table 30. Correspondence between DMAMUX registers and DMAMUX low-layer driver functions

Register	Field	Function
CxCR	DMAREQ_ID	LL_DMAMUX_GetRequestID
		LL_DMAMUX_SetRequestID

66.7 GPIO

Table 31. Correspondence between GPIO registers and GPIO low-layer driver functions

Register	Field	Function
AFRH	AFSELy	LL_GPIO_GetAFPin_8_15
		LL_GPIO_SetAFPin_8_15
AFRL	AFSELy	LL_GPIO_GetAFPin_0_7
		LL_GPIO_SetAFPin_0_7
BRR	BRy	LL_GPIO_ResetOutputPin
BSRR	BSy	LL_GPIO_SetOutputPin
IDR	IDy	LL_GPIO_IsInputPinSet
		LL_GPIO_ReadInputPort
LCKR	LCKK	LL_GPIO_IsAnyPinLocked
		LL_GPIO_LockPin
	LCKy	LL_GPIO_IsPinLocked
MODER	MODEy	LL_GPIO_GetPinMode
		LL_GPIO_SetPinMode
ODR	ODy	LL_GPIO_IsOutputPinSet
		LL_GPIO_ReadOutputPort
		LL_GPIO_TogglePin
		LL_GPIO_WriteOutputPort
OSPEEDR	OSPEEDy	LL_GPIO_GetPinSpeed
		LL_GPIO_SetPinSpeed
OTYPER	OTy	LL_GPIO_GetPinOutputType
		LL_GPIO_SetPinOutputType
PUPDR	PUPDy	LL_GPIO_GetPinPull
		LL_GPIO_SetPinPull

66.8 I2C

Table 32. Correspondence between I2C registers and I2C low-layer driver functions

Register	Field	Function
CR1	ADDRIE	LL_I2C_DisableIT_ADDR
		LL_I2C_EnableIT_ADDR
		LL_I2C_IsEnabledIT_ADDR

Register	Field	Function
CR1	ALERTEN	LL_I2C_DisableSMBusAlert
		LL_I2C_EnableSMBusAlert
		LL_I2C_IsEnabledSMBusAlert
	ANFOFF	LL_I2C_ConfigFilters
		LL_I2C_DisableAnalogFilter
		LL_I2C_EnableAnalogFilter
		LL_I2C_IsEnabledAnalogFilter
	DNF	LL_I2C_ConfigFilters
		LL_I2C_GetDigitalFilter
		LL_I2C_SetDigitalFilter
	ERRIE	LL_I2C_DisableIT_ERR
		LL_I2C_EnableIT_ERR
		LL_I2C_IsEnabledIT_ERR
	GCEN	LL_I2C_DisableGeneralCall
		LL_I2C_EnableGeneralCall
		LL_I2C_IsEnabledGeneralCall
	NACKIE	LL_I2C_DisableIT_NACK
		LL_I2C_EnableIT_NACK
		LL_I2C_IsEnabledIT_NACK
	NOSTRETCH	LL_I2C_DisableClockStretching
		LL_I2C_EnableClockStretching
		LL_I2C_IsEnabledClockStretching
	PE	LL_I2C_Disable
		LL_I2C_Enable
		LL_I2C_IsEnabled
	PECEN	LL_I2C_DisableSMBusPEC
		LL_I2C_EnableSMBusPEC
		LL_I2C_IsEnabledSMBusPEC
	RXDMAEN	LL_I2C_DisableDMAReq_RX
		LL_I2C_EnableDMAReq_RX
		LL_I2C_IsEnabledDMAReq_RX
	RXIE	LL_I2C_DisableIT_RX
		LL_I2C_EnableIT_RX
		LL_I2C_IsEnabledIT_RX
	SBC	LL_I2C_DisableSlaveByteControl
		LL_I2C_EnableSlaveByteControl
		LL_I2C_IsEnabledSlaveByteControl
	SMBDEN	LL_I2C_GetMode
		LL_I2C_SetMode
	SMBHEN	LL_I2C_GetMode
		LL_I2C_SetMode

Register	Field	Function
CR1	STOPIE	LL_I2C_DisableIT_STOP
		LL_I2C_EnableIT_STOP
		LL_I2C_IsEnabledIT_STOP
	TCIE	LL_I2C_DisableIT_TC
		LL_I2C_EnableIT_TC
		LL_I2C_IsEnabledIT_TC
	TXDMAEN	LL_I2C_DisableDMAReq_TX
		LL_I2C_EnableDMAReq_TX
		LL_I2C_IsEnabledDMAReq_TX
	TXIE	LL_I2C_DisableIT_TX
		LL_I2C_EnableIT_TX
		LL_I2C_IsEnabledIT_TX
CR2	ADD10	LL_I2C_GetMasterAddressingMode
		LL_I2C_HandleTransfer
		LL_I2C_SetMasterAddressingMode
	AUTOEND	LL_I2C_DisableAutoEndMode
		LL_I2C_EnableAutoEndMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAutoEndMode
	HEAD10R	LL_I2C_DisableAuto10BitRead
		LL_I2C_EnableAuto10BitRead
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledAuto10BitRead
	NACK	LL_I2C_AcknowledgeNextData
	NBYTES	LL_I2C_GetTransferSize
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferSize
	PECBYTE	LL_I2C_EnableSMBusPECCompare
		LL_I2C_IsEnabledSMBusPECCompare
	RD_WRN	LL_I2C_GetTransferRequest
		LL_I2C_HandleTransfer
		LL_I2C_SetTransferRequest
	RELOAD	LL_I2C_DisableReloadMode
		LL_I2C_EnableReloadMode
		LL_I2C_HandleTransfer
		LL_I2C_IsEnabledReloadMode
	SADD	LL_I2C_GetSlaveAddr
		LL_I2C_HandleTransfer
		LL_I2C_SetSlaveAddr
	START	LL_I2C_GenerateStartCondition
		LL_I2C_HandleTransfer

Register	Field	Function
CR2	STOP	LL_I2C_GenerateStopCondition
		LL_I2C_HandleTransfer
ICR	ADDRCF	LL_I2C_ClearFlag_ADDR
	ALERTCF	LL_I2C_ClearSMBusFlag_ALERT
	ARLOCF	LL_I2C_ClearFlag_ARLO
	BERRCF	LL_I2C_ClearFlag_BERR
	NACKCF	LL_I2C_ClearFlag_NACK
	OVRCF	LL_I2C_ClearFlag_OVR
	PECCF	LL_I2C_ClearSMBusFlag_PECERR
	STOPCF	LL_I2C_ClearFlag_STOP
	TIMOUTCF	LL_I2C_ClearSMBusFlag_TIMEOUT
ISR	ADDCODE	LL_I2C_GetAddressMatchCode
	ADDR	LL_I2C_IsActiveFlag_ADDR
	ALERT	LL_I2C_IsActiveSMBusFlag_ALERT
	ARLO	LL_I2C_IsActiveFlag_ARLO
	BERR	LL_I2C_IsActiveFlag_BERR
	BUSY	LL_I2C_IsActiveFlag_BUSY
	DIR	LL_I2C_GetTransferDirection
	NACKF	LL_I2C_IsActiveFlag_NACK
	OVR	LL_I2C_IsActiveFlag_OVR
	PECERR	LL_I2C_IsActiveSMBusFlag_PECERR
	RXNE	LL_I2C_IsActiveFlag_RXNE
	STOPF	LL_I2C_IsActiveFlag_STOP
	TC	LL_I2C_IsActiveFlag_TC
	TCR	LL_I2C_IsActiveFlag_TCR
	TIMEOUT	LL_I2C_IsActiveSMBusFlag_TIMEOUT
	TXE	LL_I2C_ClearFlag_TXE
		LL_I2C_IsActiveFlag_TXE
	TXIS	LL_I2C_IsActiveFlag_TXIS
OAR1	OA1	LL_I2C_SetOwnAddress1
	OA1EN	LL_I2C_DisableOwnAddress1
		LL_I2C_EnableOwnAddress1
		LL_I2C_IsEnabledOwnAddress1
	OA1MODE	LL_I2C_SetOwnAddress1
OAR2	OA2	LL_I2C_SetOwnAddress2
	OA2EN	LL_I2C_DisableOwnAddress2
		LL_I2C_EnableOwnAddress2
		LL_I2C_IsEnabledOwnAddress2
	OA2MSK	LL_I2C_SetOwnAddress2
PECR	PEC	LL_I2C_GetSMBusPEC
RXDR	RXDATA	LL_I2C_DMA_GetRegAddr

Register	Field	Function
RXDR	RXDATA	LL_I2C_ReceiveData8
TIMEOUTR	TEXTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
	TIDLE	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutAMode
		LL_I2C_SetSMBusTimeoutAMode
	TIMEOUTA	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutA
		LL_I2C_SetSMBusTimeoutA
	TIMEOUTB	LL_I2C_ConfigSMBusTimeout
		LL_I2C_GetSMBusTimeoutB
		LL_I2C_SetSMBusTimeoutB
	TIMOUTEN	LL_I2C_DisableSMBusTimeout
		LL_I2C_EnableSMBusTimeout
		LL_I2C_IsEnabledSMBusTimeout
TIMINGR	PRESC	LL_I2C_GetTimingPrescaler
	SCLDEL	LL_I2C_GetDataSetupTime
	SCLH	LL_I2C_GetClockHighPeriod
	SCLL	LL_I2C_GetClockLowPeriod
	SDADEL	LL_I2C_GetDataHoldTime
	TIMINGR	LL_I2C_SetTiming
TXDR	TXDATA	LL_I2C_DMA_GetRegAddr
		LL_I2C_TransmitData8

66.9 I2S

Table 33. **Correspondence between I2S registers and I2S low-layer driver functions**

Register	Field	Function
CR2	ERRIE	LL_I2S_DisableIT_ERR
		LL_I2S_EnableIT_ERR
		LL_I2S_IsEnabledIT_ERR
	RXDMAEN	LL_I2S_DisableDMAReq_RX
		LL_I2S_EnableDMAReq_RX
		LL_I2S_IsEnabledDMAReq_RX
	RXNEIE	LL_I2S_DisableIT_RXNE
		LL_I2S_EnableIT_RXNE
		LL_I2S_IsEnabledIT_RXNE
	TXDMAEN	LL_I2S_DisableDMAReq_TX
		LL_I2S_EnableDMAReq_TX
		LL_I2S_IsEnabledDMAReq_TX

Register	Field	Function
CR2	TXEIE	LL_I2S_DisableIT_TXE
		LL_I2S_EnableIT_TXE
		LL_I2S_IsEnabledIT_TXE
DR	DR	LL_I2S_ReceiveData16
		LL_I2S_TransmitData16
I2SCFGR	ASTRTEN	LL_I2S_DisableAsyncStart
		LL_I2S_EnableAsyncStart
		LL_I2S_IsEnabledAsyncStart
	CHLEN	LL_I2S_GetDataFormat
		LL_I2S_SetDataFormat
	CKPOL	LL_I2S_GetClockPolarity
		LL_I2S_SetClockPolarity
	DATLEN	LL_I2S_GetDataFormat
		LL_I2S_SetDataFormat
	I2SCFG	LL_I2S_GetTransferMode
		LL_I2S_SetTransferMode
	I2SE	LL_I2S_Disable
		LL_I2S_Enable
		LL_I2S_IsEnabled
	I2SMOD	LL_I2S_Enable
	I2SSTD	LL_I2S_GetStandard
		LL_I2S_SetStandard
	PCMSYNC	LL_I2S_GetStandard
		LL_I2S_SetStandard
I2SPR	I2SDIV	LL_I2S_GetPrescalerLinear
		LL_I2S_SetPrescalerLinear
	MCKOE	LL_I2S_DisableMasterClock
		LL_I2S_EnableMasterClock
		LL_I2S_IsEnabledMasterClock
	ODD	LL_I2S_GetPrescalerParity
		LL_I2S_SetPrescalerParity
SR	BSY	LL_I2S_IsActiveFlag_BSY
	CHSIDE	LL_I2S_IsActiveFlag_CHSIDE
	FRE	LL_I2S_ClearFlag_FRE
		LL_I2S_IsActiveFlag_FRE
	OVR	LL_I2S_ClearFlag_OVR
		LL_I2S_IsActiveFlag_OVR
	RXNE	LL_I2S_IsActiveFlag_RXNE
	TXE	LL_I2S_IsActiveFlag_TXE
	UDR	LL_I2S_ClearFlag_UDR
		LL_I2S_IsActiveFlag_UDR

66.10 IWDG

Table 34. Correspondence between IWDG registers and IWDG low-layer driver functions

Register	Field	Function
KR	KEY	LL_IWDG_DisableWriteAccess
		LL_IWDG_Enable
		LL_IWDG_EnableWriteAccess
		LL_IWDG_ReloadCounter
PR	PR	LL_IWDG_GetPrescaler
		LL_IWDG_SetPrescaler
RLR	RL	LL_IWDG_GetReloadCounter
		LL_IWDG_SetReloadCounter
SR	PVU	LL_IWDG_IsActiveFlag_PVU
		LL_IWDG_IsReady
	RVU	LL_IWDG_IsActiveFlag_RVU
		LL_IWDG_IsReady
	WVU	LL_IWDG_IsActiveFlag_WVU
		LL_IWDG_IsReady
WINR	WIN	LL_IWDG_GetWindow
		LL_IWDG_SetWindow

66.11 LPUART

Table 35. Correspondence between LPUART registers and LPUART low-layer driver functions

Register	Field	Function
BRR	BRR	LL_LPUART_GetBaudRate
		LL_LPUART_SetBaudRate
CR1	CMIE	LL_LPUART_DisableIT_CM
		LL_LPUART_EnableIT_CM
		LL_LPUART_IsEnabledIT_CM
	DEAT	LL_LPUART_GetDEAssertionTime
		LL_LPUART_SetDEAssertionTime
	DEDT	LL_LPUART_GetDEDeassertionTime
		LL_LPUART_SetDEDeassertionTime
	FIFOEN	LL_LPUART_DisableFIFO
		LL_LPUART_EnableFIFO
		LL_LPUART_IsEnabledFIFO
	IDLEIE	LL_LPUART_DisableIT_IDLE
		LL_LPUART_EnableIT_IDLE
		LL_LPUART_IsEnabledIT_IDLE
	M	LL_LPUART_ConfigCharacter
		LL_LPUART_GetDataWidth

Register	Field	Function
CR1	M	LL_LPUART_SetDataWidth
	MME	LL_LPUART_DisableMuteMode
		LL_LPUART_EnableMuteMode
		LL_LPUART_IsEnabledMuteMode
	PCE	LL_LPUART_ConfigCharacter
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	PEIE	LL_LPUART_DisableIT_PE
		LL_LPUART_EnableIT_PE
		LL_LPUART_IsEnabledIT_PE
	PS	LL_LPUART_ConfigCharacter
		LL_LPUART_GetParity
		LL_LPUART_SetParity
	RE	LL_LPUART_DisableDirectionRx
		LL_LPUART_EnableDirectionRx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	RXFFIE	LL_LPUART_DisableIT_RXFF
		LL_LPUART_EnableIT_RXFF
		LL_LPUART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	LL_LPUART_DisableIT_RXNE_RXFNE
		LL_LPUART_EnableIT_RXNE_RXFNE
		LL_LPUART_IsEnabledIT_RXNE_RXFNE
	TCIE	LL_LPUART_DisableIT_TC
		LL_LPUART_EnableIT_TC
		LL_LPUART_IsEnabledIT_TC
	TE	LL_LPUART_DisableDirectionTx
		LL_LPUART_EnableDirectionTx
		LL_LPUART_GetTransferDirection
		LL_LPUART_SetTransferDirection
	TXEIE_TXFNFIE	LL_LPUART_DisableIT_TXE_TXFNF
		LL_LPUART_EnableIT_TXE_TXFNF
		LL_LPUART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_LPUART_DisableIT_TXFE
		LL_LPUART_EnableIT_TXFE
		LL_LPUART_IsEnabledIT_TXFE
	UE	LL_LPUART_Disable
		LL_LPUART_Enable
		LL_LPUART_IsEnabled
	UESM	LL_LPUART_DisableInStopMode
		LL_LPUART_EnableInStopMode

Register	Field	Function
CR1	UESM	LL_LPUART_IsEnabledInStopMode
	WAKE	LL_LPUART_GetWakeUpMethod
		LL_LPUART_SetWakeUpMethod
CR2	ADD	LL_LPUART_ConfigNodeAddress
		LL_LPUART_GetNodeAddress
	ADDM7	LL_LPUART_ConfigNodeAddress
		LL_LPUART_GetNodeAddressLen
	DATAINV	LL_LPUART_GetBinaryDataLogic
		LL_LPUART_SetBinaryDataLogic
	MSBFIRST	LL_LPUART_GetTransferBitOrder
		LL_LPUART_SetTransferBitOrder
	RXINV	LL_LPUART_GetRXPinLevel
		LL_LPUART_SetRXPinLevel
	STOP	LL_LPUART_ConfigCharacter
		LL_LPUART_GetStopBitsLength
		LL_LPUART_SetStopBitsLength
	SWAP	LL_LPUART_GetTXRXSwap
		LL_LPUART_SetTXRXSwap
	TXINV	LL_LPUART_GetTXPinLevel
		LL_LPUART_SetTXPinLevel
CR3	CTSE	LL_LPUART_DisableCTSHWFlowCtrl
		LL_LPUART_EnableCTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	CTSIE	LL_LPUART_DisableIT_CTS
		LL_LPUART_EnableIT_CTS
		LL_LPUART_IsEnabledIT_CTS
	DDRE	LL_LPUART_DisableDMADeactOnRxErr
		LL_LPUART_EnableDMADeactOnRxErr
		LL_LPUART_IsEnabledDMADeactOnRxErr
	DEM	LL_LPUART_DisableDEMode
		LL_LPUART_EnableDEMode
		LL_LPUART_IsEnabledDEMode
	DEP	LL_LPUART_GetDESignalPolarity
		LL_LPUART_SetDESignalPolarity
	DMAR	LL_LPUART_DisableDMAReq_RX
		LL_LPUART_EnableDMAReq_RX
		LL_LPUART_IsEnabledDMAReq_RX
	DMAT	LL_LPUART_DisableDMAReq_TX
		LL_LPUART_EnableDMAReq_TX
		LL_LPUART_IsEnabledDMAReq_TX

Register	Field	Function
CR3	EIE	LL_LPUART_DisableIT_ERROR
		LL_LPUART_EnableIT_ERROR
		LL_LPUART_IsEnabledIT_ERROR
	HDSEL	LL_LPUART_DisableHalfDuplex
		LL_LPUART_EnableHalfDuplex
		LL_LPUART_IsEnabledHalfDuplex
	OVRDIS	LL_LPUART_DisableOverrunDetect
		LL_LPUART_EnableOverrunDetect
		LL_LPUART_IsEnabledOverrunDetect
	RTSE	LL_LPUART_DisableRTSHWFlowCtrl
		LL_LPUART_EnableRTSHWFlowCtrl
		LL_LPUART_GetHWFlowCtrl
		LL_LPUART_SetHWFlowCtrl
	RXFTCFG	LL_LPUART_ConfigFIFOsThreshold
		LL_LPUART_GetRXFIFOThreshold
		LL_LPUART_SetRXFIFOThreshold
	RXFTIE	LL_LPUART_DisableIT_RXFT
		LL_LPUART_EnableIT_RXFT
		LL_LPUART_IsEnabledIT_RXFT
	TXFTCFG	LL_LPUART_ConfigFIFOsThreshold
		LL_LPUART_GetTXFIFOThreshold
		LL_LPUART_SetTXFIFOThreshold
	TXFTIE	LL_LPUART_DisableIT_TXFT
		LL_LPUART_EnableIT_TXFT
		LL_LPUART_IsEnabledIT_TXFT
	WUFIE	LL_LPUART_DisableIT_WKUP
		LL_LPUART_EnableIT_WKUP
		LL_LPUART_IsEnabledIT_WKUP
	WUS	LL_LPUART_GetWKUPType
		LL_LPUART_SetWKUPType
ICR	CMCF	LL_LPUART_ClearFlag_CM
	CTSCF	LL_LPUART_ClearFlag_nCTS
	FECF	LL_LPUART_ClearFlag_FE
	IDLECF	LL_LPUART_ClearFlag_IDLE
	NECF	LL_LPUART_ClearFlag_NE
	ORECF	LL_LPUART_ClearFlag_ORE
	PECF	LL_LPUART_ClearFlag_PE
	TCCF	LL_LPUART_ClearFlag_TC
	WUCF	LL_LPUART_ClearFlag_WKUP
ISR	BUSY	LL_LPUART_IsActiveFlag_BUSY
	CMF	LL_LPUART_IsActiveFlag_CM

Register	Field	Function
ISR	CTS	LL_LPUART_IsActiveFlag_CTS
	CTSIF	LL_LPUART_IsActiveFlag_nCTS
	FE	LL_LPUART_IsActiveFlag_FE
	IDLE	LL_LPUART_IsActiveFlag_IDLE
	NE	LL_LPUART_IsActiveFlag_NE
	ORE	LL_LPUART_IsActiveFlag_ORE
	PE	LL_LPUART_IsActiveFlag_PE
	REACK	LL_LPUART_IsActiveFlag_REACK
	RWU	LL_LPUART_IsActiveFlag_RWU
	RXFF	LL_LPUART_IsActiveFlag_RXFF
	RXFT	LL_LPUART_IsActiveFlag_RXFT
	RXNE_RXFNE	LL_LPUART_IsActiveFlag_RXNE_RXFNE
	SBKF	LL_LPUART_IsActiveFlag_SBK
	TC	LL_LPUART_IsActiveFlag_TC
	TEACK	LL_LPUART_IsActiveFlag_TEACK
	TXE_TXFNF	LL_LPUART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_LPUART_IsActiveFlag_TXFE
	TXFT	LL_LPUART_IsActiveFlag_TXFT
	WUF	LL_LPUART_IsActiveFlag_WKUP
PRESC	PRESCALER	LL_LPUART_GetPrescaler
		LL_LPUART_SetPrescaler
RDR	RDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_ReceiveData8
		LL_LPUART_ReceiveData9
RQR	MMRQ	LL_LPUART_RequestEnterMuteMode
	RXFRQ	LL_LPUART_RequestRxDataFlush
	SBKRQ	LL_LPUART_RequestBreakSending
	TXFRQ	LL_LPUART_RequestTxDataFlush
TDR	TDR	LL_LPUART_DMA_GetRegAddr
		LL_LPUART_TransmitData8
		LL_LPUART_TransmitData9

66.12 PWR

Table 36. Correspondence between PWR registers and PWR low-layer driver functions

Register	Field	Function
CR1	APC	LL_PWR_DisablePUPDCfg
		LL_PWR_EnablePUPDCfg
		LL_PWR_IsEnabledPUPDCfg
	ENSDNBOR	LL_PWR_DisableBORinSDN
		LL_PWR_EnableBORinSDN

Register	Field	Function
CR1	ENSDNBOR	LL_PWR_IsEnabledBORinSDN
	LPMS	LL_PWR_GetPowerMode
		LL_PWR_SetPowerMode
CR2	PVDE	LL_PWR_DisablePVD
		LL_PWR_EnablePVD
		LL_PWR_IsEnabledPVD
	PVDLS	LL_PWR_GetPVDLevel
		LL_PWR_SetPVDLevel
CR3	EIWL	LL_PWR_DisableInternWU
		LL_PWR_EnableInternWU
		LL_PWR_IsEnabledInternWU
	EIWL2	LL_PWR_EnableInternWU2
	EWBLE	LL_PWR_DisableWU_EWBLE
		LL_PWR_EnableWU_EWBLE
		LL_PWR_IsEnabledWU_EWBLE
	EWBLEHCPU	LL_PWR_DisableWU_EWBLEHCPU
		LL_PWR_EnableWU_EWBLEHCPU
		LL_PWR_IsEnabledWU_EWBLEHCPU
	EWU0	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU1	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU10	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU11	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU2	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU3	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU4	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU5	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin

Register	Field	Function
CR3	EWU5	LL_PWR_IsEnabledWakeUpPin
	EWU6	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU7	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU8	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU9	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
CR4	WUP0	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP1	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP10	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP11	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP2	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP3	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP4	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP5	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP6	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP7	LL_PWR_IsWakeUpPinPolarityLow

Register	Field	Function
CR4	WUP7	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP8	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP9	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
CR5	NOSMPS	LL_PWR_GetSMPSMode
		LL_PWR_SetSMPSMode
	SMPSFBYP	LL_PWR_IsEnabledSMPSPrechargeMode
		LL_PWR_SetSMPSPrechargeMode
	SMPSLPOPEN	LL_PWR_SetSMPSOpenMode
	SMPSLVL	LL_PWR_SMPS_GetOutputVoltageLevel
		LL_PWR_SMPS_SetOutputVoltageLevel
CR6	EWU12	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU13	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU14	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU15	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU16	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU17	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU18	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU19	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU20	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin

Register	Field	Function
CR6	EWU20	LL_PWR_IsEnabledWakeUpPin
	EWU21	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU22	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU23	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU24	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU25	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU26	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
	EWU27	LL_PWR_DisableWakeUpPin
		LL_PWR_EnableWakeUpPin
		LL_PWR_IsEnabledWakeUpPin
CR7	WUP12	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP13	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP14	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP15	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP16	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP17	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP18	LL_PWR_IsWakeUpPinPolarityLow

Register	Field	Function
CR7	WUP18	LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
	WUP19	LL_PWR_IsWakeUpPinPolarityLow
		LL_PWR_SetWakeUpPinPolarityHigh
		LL_PWR_SetWakeUpPinPolarityLow
SR1	BHWF	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	IWUF	LL_PWR_IsActiveFlag_InternWU
	IWUF2	LL_PWR_IsActiveFlag_InternWU2
	WBLEF	LL_PWR_ClearFlag_WBLEF
		Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WBLEF
	WBLEHCPUF	Section 56.1.1: LL_PWR_ClearFlag_WBLEHCPU
		LL_PWR_IsActiveFlag_WBLEHCPUF
	WUF0	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU0
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU0
	WUF1	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU1
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU1
	WUF10	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU10
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU10
	WUF11	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU11
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU11
	WUF2	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU2
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU2
	WUF3	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU3
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU3
	WUF4	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU4

Register	Field	Function
SR1	WUF4	LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU4
	WUF5	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU5
		LL_PWR_ClearFlag_WU7
		LL_PWR_ClearFlag_WU8
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU5
	WUF6	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU6
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU6
	WUF7	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU7
	WUF8	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU8
	WUF9	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU9
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU9
	WUFI	LL_PWR_GetWakeupSource
	WUFI2	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
SR2	SMPSBYPR	LL_PWR_IsSMPSinPRECHARGEMode
	SMPSENR	LL_PWR_IsSMPSinRUNMode
	SMPSRDY	LL_PWR_IsSMPSReady
SR3	IWUF2	Section 56.1.1: LL_PWR_ClearFlag_BORH
	WUF12	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU12
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU12
	WUF13	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU13
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU13
	WUF14	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU14
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU14

Register	Field	Function
SR3	WUF15	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU15
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU15
	WUF16	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU16
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU16
	WUF17	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU17
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU17
	WUF18	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU18
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU18
	WUF19	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_ClearFlag_WU19
		LL_PWR_GetWakeupSource
		LL_PWR_IsActiveFlag_WU19
	WUF20	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF21	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF22	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF23	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF24	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF25	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF26	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource
	WUF27	Section 56.1.1: LL_PWR_ClearFlag_WU
		LL_PWR_GetWakeupSource

66.13 RADIO

Table 37. Correspondence between RADIO registers and RADIO low-layer driver functions

Register	Field	Function
BLUE_SLEEP_REQUEST_MODE	BLE_WAKEUP_EN	LL_RADIO_TIMER_DisableBLEWakeupTimer
		LL_RADIO_TIMER_EnableBLEWakeupTimer
		LL_RADIO_TIMER_IsEnabledBLEWakeupTimer
	FORCE_SLEEPING	LL_RADIO_TIMER_DisableBLEWakeupTimerForceSleeping
		LL_RADIO_TIMER_EnableBLEWakeupTimerForceSleeping
		LL_RADIO_TIMER_IsEnabledBLEWakeupTimerForceSleeping
	SLEEP_EN	LL_RADIO_TIMER_DisableWakeupTimerLowPowerMode
		LL_RADIO_TIMER_EnableWakeupTimerLowPowerMode
		LL_RADIO_TIMER_IsEnabledWakeupTimerLowPowerMode
CM0_SLEEP_REQUEST_MODE	CPU_WAKEUP_EN	LL_RADIO_TIMER_DisableCPUWakeupTimer
		LL_RADIO_TIMER_EnableCPUWakeupTimer
		LL_RADIO_TIMER_IsEnabledCPUWakeupTimer
	FORCE_SLEEPING	LL_RADIO_TIMER_DisableCPUWakeupTimerForceSleeping
		LL_RADIO_TIMER_EnableCPUWakeupTimerForceSleeping
		LL_RADIO_TIMER_IsEnabledCPUWakeupTimerForceSleeping
	SLEEP_REQ_MODE	LL_RADIO_TIMER_GetSleepRequestMode
		LL_RADIO_TIMER_SetSleepRequestMode
CMDREG	BLUE_CMDREG_TXRXSKIP	LL_RADIO_BlueSetTxRxSkip
	CLEARSEMAREQ	LL_RADIO_BlueGetClearSemaphoreRequest
		LL_RADIO_BlueSetClearSemaphoreRequest
INTERRUPT1LATENCYREG	INTERRUPT1LATENCY	LL_RADIO_BlueGetInterrupt1Latency
INTERRUPT1REG	ALL	LL_RADIO_BlueGetInterrupt1RegRegister
		LL_RADIO_BlueSetInterrupt1RegRegister
MANAESCMDREG	START	LL_RADIO_BlueGetManAESCmdStart
		LL_RADIO_BlueSetManAESCmdStart
MANAESKEY0REG	MANAESKEY_31_0	LL_RADIO_BlueGetManualAesKey_31_0
		LL_RADIO_BlueSetManualAesKey_31_0
MANAESSTATREG	BUSY	LL_RADIO_BlueGetManAESStatusBusy
STATUS2REG	ANTENNA_SWITCHING_PATTERN_ACCESS_ERROR	LL_RADIO_GetAntennaSwitchingPatternAccessError
	ANTENNA_SWITCHING_PATTERN_ADDRESS_ERROR	LL_RADIO_GetAntennaSwitchingPatternAddressError
	IQSAMPLES_MISSING_ERROR	LL_RADIO_GetIQSamplesMissingError
	IQSAMPLES_NUMBER	LL_RADIO_GetIQSamplesNumber
	IQSAMPLES_READY	LL_RADIO_GetIQSamplesReady
STATUSREG	ACTIVE2ERROR	LL_RADIO_BlueGetActive2Error
	TIMECAPTURETRIG	LL_RADIO_TIMER_IsActiveFlag_TIMECAPTURETRIG
TIMEOUTDESTREG	DESTINATION	LL_RADIO_TIMER_DisableTimer1

Register	Field	Function
TIMEOUTDESTREG	DESTINATION	LL_RADIO_TIMER_DisableTimer2
		LL_RADIO_TIMER_EnableTimer1
		LL_RADIO_TIMER_EnableTimer2
		LL_RADIO_TIMER_IsEnabledTimer1
		LL_RADIO_TIMER_IsEnabledTimer2
TIMERCAPTUREREG	TIMERCAPTURE	LL_RADIO_BlueGetTimerCapture
WAKEUP_BLE_IRQ_ ENABLE	WAKEUP_IT	LL_RADIO_TIMER_DisableBLEWakeupIT
		LL_RADIO_TIMER_EnableBLEWakeupIT
		LL_RADIO_TIMER_IsEnabledBLEWakeupIT
WAKEUP_BLE_IRQ_ STATUS	WAKEUP_IT	LL_RADIO_TIMER_ClearFlag_BLEWakeup
		LL_RADIO_TIMER_IsActiveFlag_BLEWakeup
WAKEUP_CM0_IRQ_ ENABLE	WAKEUP_IT	LL_RADIO_TIMER_DisableCPUWakeupIT
		LL_RADIO_TIMER_EnableCPUWakeupIT
WAKEUP_CM0_IRQ_ STATUS	WAKEUP_IT	LL_RADIO_TIMER_ClearFlag_CPUWakeup
		LL_RADIO_TIMER_IsActiveFlag_CPUWakeup
WAKEUP_CPU_IRQ_ ENABLE	WAKEUP_IT	LL_RADIO_TIMER_IsEnabledCPUWakeupIT
WORD0	ENCRYPTON	LL_RADIO_Encryption_Disable
		LL_RADIO_Encryption_Enable
		LL_RADIO_Encryption_IsEnabled
	NESN	LL_RADIO_GetNextExpectedSequenceNumber
		LL_RADIO_SetNextExpectedSequenceNumber
		LL_RADIO_ToggleNextExpectedSequenceNumber
	NEXTPTR	LL_RADIO_GetNextPointer
		LL_RADIO_SetNextPointer
	RADIOCOMLISTENA	LL_RADIO_RadioCommandList_Disable
		LL_RADIO_RadioCommandList_Enable
		LL_RADIO_RadioCommandList_IsEnabled
	RADIOCONFIGPTR	LL_RADIO_GetRadioConfigurationAddressPointer
		LL_RADIO_SetRadioConfigurationAddressPointer
	REMAP_CHAN	LL_RADIO_GetRemapChannel
		LL_RADIO_SetRemapChannel
	RXPHY	LL_RADIO_GetReceptionPhy
		LL_RADIO_SetReceptionPhy
	SN	LL_RADIO_GetSequenceNumber
		LL_RADIO_SetSequenceNumber
		LL_RADIO_ToggleSequenceNumber
	TXMODE	LL_RADIO_TxMode_Disable
		LL_RADIO_TxMode_Enable
		LL_RADIO_TxMode_IsEnabled
	TXPHY	LL_RADIO_GetTransmissionPhy
		LL_RADIO_SetTransmissionPhy

Register	Field	Function
WORD0	UCHAN	LL_RADIO_GetUnmappedChannel
		LL_RADIO_SetUnmappedChannel
WORD1	ACTIVE	LL_RADIO_GlobalDisableBlue
		LL_RADIO_GlobalEnableBlue
		LL_RADIO_IsEnabledGlobalBlue
	ADVERTISE	LL_RADIO_GetAdvertise
		LL_RADIO_SetAdvertise
	ALLTABLEREADY	LL_RADIO_GetAllTableDataReady
		LL_RADIO_SetAllTableDataReady
	CALREQ	LL_RADIO_GetCalibrationRequest
		LL_RADIO_SetCalibrationRequest
	CRCINITSEL	LL_RADIO_GetCRCInitializationSelector
		LL_RADIO_SetCRCInitializationSelector
	CURSTMACHNUM	LL_RADIO_GetCurrentConnectionMachineNumber
		LL_RADIO_SetCurrentConnectionMachineNumber
	DISABLEWHITENING	LL_RADIO_GetDisableWhitening
		LL_RADIO_SetDisableWhitening
	INCCHAN	LL_RADIO_IncrementerChannel_Disable
		LL_RADIO_IncrementerChannel_Enable
		LL_RADIO_IncrementerChannel_IsEnabled
	KEEPSEMAREQ	LL_RADIO_GetKeepSemaRequest
		LL_RADIO_SetKeepSemaRequest
	NEXTTXMODE	LL_RADIO_GetNextTxMode
		LL_RADIO_SetNextTxMode
	SN_EN	LL_RADIO_AutomaticSnNesnHardwareMechanism_Disable
		LL_RADIO_AutomaticSnNesnHardwareMechanism_Enable
		LL_RADIO_AutomaticSnNesnHardwareMechanism_IsEnabled
	TIMER12INITDELAYCAL	LL_RADIO_GetTimer12InitDelayCal
		LL_RADIO_SetTimer12InitDelayCal
	TIMER2INITDELAYNOCAL	LL_RADIO_GetTimer12InitDelayNoCal
		LL_RADIO_SetTimer12InitDelayNoCal
	TXDATAREADY	LL_RADIO_GetTransmissionDataReady
		LL_RADIO_SetTransmissionDataReady
	TXPOINT	LL_RADIO_GetTxPoint
		LL_RADIO_SetTxPoint
	WAKEUPINITDELAY	LL_RADIO_GetWakeupInitDelay
		LL_RADIO_SetWakeupInitDelay
WORD2	DATAPTR	LL_RADIO_GetDataPointer
		LL_RADIO_SetDataPointer
	RCVPOINT	LL_RADIO_GetRcvPoint
		LL_RADIO_SetRcvPoint

Register	Field	Function
WORD2	RECEIVECALDELAYCHK	LL_RADIO_GetReceivedCalDelayChk
		LL_RADIO_SetReceivedCalDelayChk
	RECEIVENOCALDELAYCHK	LL_RADIO_GetReceivedNoCalDelayChk
		LL_RADIO_SetReceivedNoCalDelayChk
	TRANSMITCALDELAYCHK	LL_RADIO_GetTransmitCalDelayChk
		LL_RADIO_SetTransmitCalDelayChk
	TRANSMITNOCALDELAYCHK	LL_RADIO_GetTransmitNoCalDelayChk
		LL_RADIO_SetTransmitNoCalDelayChk
WORD3	CONFIGENDDURATION	LL_RADIO_GetConfigurationEndDuration
		LL_RADIO_SetConfigurationEndDuration
	INTDONE	LL_RADIO_IntDone_Disable
		LL_RADIO_IntDone_Enable
		LL_RADIO_IntDone_IsEnabled
	INTRCVCRCERR	LL_RADIO_IntRcvCRCErr_Disable
		LL_RADIO_IntRcvCRCErr_Enable
		LL_RADIO_IntRcvCRCErr_IsEnabled
	INTRCVOK	LL_RADIO_IntRcvOk_Disable
		LL_RADIO_IntRcvOk_Enable
		LL_RADIO_IntRcvOk_IsEnabled
	INTRCVTIMEOUT	LL_RADIO_IntRcvTimeout_Disable
		LL_RADIO_IntRcvTimeout_Enable
		LL_RADIO_IntRcvTimeout_IsEnabled
	INTTIMECAPTURE	LL_RADIO_GetIntTimeCapture
		LL_RADIO_SetIntTimeCapture
	INTTXOK	LL_RADIO_GetIntTxOk
		LL_RADIO_SetIntTxOk
	TIMECAPTURE	LL_RADIO_GetTimeCapture
		LL_RADIO_SetTimeCapture
	TIMECAPTURESEL	LL_RADIO_GetTimeCaptureSel
		LL_RADIO_SetTimeCaptureSel
	TIMER2	LL_RADIO_GetTimer2Triggering
		LL_RADIO_SetTimer2Triggering
	TIMER2EN	LL_RADIO_Timer2_Disable
		LL_RADIO_Timer2_Enable
		LL_RADIO_Timer2_IsEnabled
	TRIGDONE	LL_RADIO_GetTrigDone
		LL_RADIO_SetTrigDone
	TRIGRCV	LL_RADIO_GetTrigRcv
		LL_RADIO_SetTrigRcv
	TXDATAREADYCHECK	LL_RADIO_GetTxDataReadyCheck
		LL_RADIO_SetTxDataReadyCheck

Register	Field	Function
WORD3	TXDELAYEND	LL_RADIO_GetTxDelayEnd
		LL_RADIO_SetTxDelayEnd
	TXDELAYSTART	LL_RADIO_GetTxDelayStart
		LL_RADIO_SetTxDelayStart
	TXPOINTPREV	LL_RADIO_GetTxPointPrevious
		LL_RADIO_SetTxPointPrevious
WORD4	RCVPOINTPREV	LL_RADIO_GetRcvPointPrevious
		LL_RADIO_SetRcvPointPrevious
	RCVTIMEOUT	LL_RADIO_GetReceiveWindowTimeout
		LL_RADIO_SetReceiveWindowTimeout
	TXREADYTIMEOUT	LL_RADIO_GetTransmissionReadyTimeout
		LL_RADIO_SetTransmissionReadyTimeout
WORD5	CHKFLAGAUTOCLEARENA	LL_RADIO_ChkFlagAutoclearEnable_Disable
		LL_RADIO_ChkFlagAutoclearEnable_Enable
		LL_RADIO_ChkFlagAutoclearEnable_IsEnabled
	INTACTIVE2ERR	LL_RADIO_Active2ErrorInterrupt_Disable
		LL_RADIO_Active2ErrorInterrupt_Enable
		LL_RADIO_Active2ErrorInterrupt_IsEnabled
	INTNOACTIVELERROR	LL_RADIO_NoActiveLErrorInterrupt_Disable
		LL_RADIO_NoActiveLErrorInterrupt_Enable
		LL_RADIO_NoActiveLErrorInterrupt_IsEnabled
	INTTXRXSKIP	LL_RADIO_TxRxSkipInterrupt_Disable
		LL_RADIO_TxRxSkipInterrupt_Enable
		LL_RADIO_TxRxSkipInterrupt_IsEnabled
WORD6	PCNTTX_31_0	LL_RADIO_GetPacketCounterTx_31_0
		LL_RADIO_SetPacketCounterTx_31_0
WORD7	PCNTRCV_23_0	LL_RADIO_GetPacketCounterRx_23_0
		LL_RADIO_SetPacketCounterRx_23_0
	PCNTTX_39_32	LL_RADIO_GetPacketCounterTx_39_32
		LL_RADIO_SetPacketCounterTx_39_32
WORD8	INTENCERROR	LL_RADIO_ReceiveEncryptionErrorInterrupt_Disable
		LL_RADIO_ReceiveEncryptionErrorInterrupt_Enable
	PCNTRCV_39_24	LL_RADIO_GetPacketCounterRx_39_24
		LL_RADIO_SetPacketCounterRx_39_24
WORD9	ACCADDR	LL_RADIO_GetAccessAddress
		LL_RADIO_SetAccessAddress
WORDA	CRCINIT	LL_RADIO_GetCRCInitializationValue
		LL_RADIO_SetCRCInitializationValue
	MAXRECEIVEDLENGTH	LL_RADIO_GetMaximumReceiveLength
		LL_RADIO_SetMaximumReceiveLength
WORDB	HOPINCR	LL_RADIO_GetHopIncrement

Register	Field	Function
WORDB	HOPINCR	LL_RADIO_SetHopIncrement
	PAPOWER	LL_RADIO_GetPAPower
		LL_RADIO_SetPAPower
	USEDCHANNELFLAGS_15_0	LL_RADIO_GetUsedChannelFlags_15_0
		LL_RADIO_SetUsedChannelFlags_15_0
WORDC	USEDCHANNELFLAGS_36_16	LL_RADIO_GetUsedChannelFlags_36_16
		LL_RADIO_SetUsedChannelFlags_36_16
WORDE	STATEMACH_WORDE_ENCRYPTIV_0_31	LL_RADIO_GetEncryptIV
		LL_RADIO_SetEncryptIV

66.14 RCC

Table 38. Correspondence between RCC registers and RCC low-layer driver functions

Register	Field	Function
APB2ENR	CLKBLEDIV	Section 59.2.1: LL_RCC_GetRFClockSource
		Section 59.2.1: LL_RCC_SetRFClockSource
APB2RSTR	MRBLERST	LL_RCC_IsRFUnderReset
CFGR	CCOPRE	LL_RCC_ConfigMCO
	CLKSLOWSEL	LL_RCC_LSCO_GetSource
		LL_RCC_LSCO_SetSource
	CLKSYSDIV	LL_RCC_GetDirectHSEPrescaler
		LL_RCC_GetRC64MPLLPrescaler
		LL_RCC_SetDirectHSEPrescaler
		LL_RCC_SetRC64MPLLPrescaler
	CLKSYSDIV_STATUS	LL_RCC_GetCLKSYSPrescalerStatus
	HSESEL/STOPHSI	LL_RCC_DIRECT_HSE_IsEnabled
		LL_RCC_HSE_Disable
		LL_RCC_HSE_Enable
	HSESEL_STATUS	LL_RCC_Get_DIRECT_HSESEL_Status
	IOBOOSTCLKEN	LL_RCC_IOBOOSTCLK_Disable
		LL_RCC_IOBOOSTCLK_Enable
		LL_RCC_IOBOOSTCLK_IsEnabled
	IOBOOSTEN	LL_RCC_IOBOOST_Disable
		LL_RCC_IOBOOST_Enable
		LL_RCC_IOBOOST_IsEnabled
	LCOEN	LL_RCC_LSCOinDeepStop_Disable
		LL_RCC_LSCOinDeepStop_Enable
		LL_RCC_LSCOinDeepStop_IsEnabled
	LCOSEL	LL_RCC_ConfigLSCO
	MCOSEL	LL_RCC_ConfigMCO
	SMPSDIV	LL_RCC_GetSMPSPrescaler

Register	Field	Function
CFGR	SMPSDIV	LL_RCC_SetSMPSPrescaler
	SPI3I2SCLKSEL	LL_RCC_GetSPI3I2SClockSource
		LL_RCC_SetSPI3I2SClockSource
	STOPHSI	LL_RCC_HSI_Disable
		LL_RCC_HSI_Enable
CIER	HSERDYIE	LL_RCC_DisableIT_HSERDY
		LL_RCC_EnableIT_HSERDY
		LL_RCC_IsEnabledIT_HSERDY
	HSIPLLRDYIE	LL_RCC_DisableIT_RC64MPLLRDY
		LL_RCC_EnableIT_RC64MPLLRDY
		LL_RCC_IsEnabledIT_RC64MPLLRDY
	HSIRDYIE	LL_RCC_DisableIT_HSIRDY
		LL_RCC_EnableIT_HSIRDY
		LL_RCC_IsEnabledIT_HSIRDY
	LSERDYIE	LL_RCC_DisableIT_LSERDY
		LL_RCC_EnableIT_LSERDY
		LL_RCC_IsEnabledIT_LSERDY
	LSIRDYIE	LL_RCC_DisableIT_LSIRDY
		LL_RCC_EnableIT_LSIRDY
		LL_RCC_IsEnabledIT_LSIRDY
	RCC_CIER_HSIPLLUNLOCKDETIE	LL_RCC_DisableIT_RC64MPLLUNLOCKDET
		LL_RCC_EnableIT_RC64MPLLUNLOCKDET
		LL_RCC_IsEnabledIT_RC64MPLLUNLOCKDET
	RCC_CIER_LPURSTIE	LL_RCC_DisableIT_LPURSTREL
		LL_RCC_EnableIT_LPURSTREL
		LL_RCC_IsEnabledIT_LPURSTREL
	RCC_CIER_RTCRSTIE	LL_RCC_DisableIT_RTCRSTREL
		LL_RCC_EnableIT_RTCRSTREL
		LL_RCC_IsEnabledIT_RTCRSTREL
	RCC_CIER_WDGRSTIE	LL_RCC_DisableIT_WDGRSTREL
		LL_RCC_EnableIT_WDGRSTREL
		LL_RCC_IsEnabledIT_WDGRSTREL
CIFR	HSERDYF	LL_RCC_ClearFlag_HSERDY
		LL_RCC_IsActiveFlag_HSERDY
	HSIPLLRDYF	LL_RCC_ClearFlag_RC64MPLLRDY
		LL_RCC_IsActiveFlag_RC64MPLLRDY
	HSIPLLUNLOCKDETF	LL_RCC_ClearFlag_RC64MPLLUNLOCKDET
		LL_RCC_IsActiveFlag_RC64MPLLUNLOCKDET
	HSIRDYF	LL_RCC_ClearFlag_HSIRDY
		LL_RCC_IsActiveFlag_HSIRDY
	LPURSTF	LL_RCC_ClearFlag_LPURSTREL

Register	Field	Function
CIFR	LPURSTF	LL_RCC_IsActiveFlag_LPURSTREL
	LSERDYF	LL_RCC_ClearFlag_LSERDY
		LL_RCC_IsActiveFlag_LSERDY
	LSIRDYF	LL_RCC_ClearFlag_LSIRDY
		LL_RCC_IsActiveFlag_LSIRDY
	RTCRSTF	Section 59.2.1: LL_RCC_ClearFlag_RTCRRSTREL
		LL_RCC_IsActiveFlag_RTCRSTREL
	WDGRSTF	LL_RCC_ClearFlag_WDGRSTREL
		LL_RCC_IsActiveFlag_WDGRSTREL
CR	HSEON	LL_RCC_GetSysClkSource
		LL_RCC_HSE_Disable
		LL_RCC_HSE_Enable
		LL_RCC_HSE_IsEnabled
		LL_RCC_SetSysClkSource
	HSERDY	LL_RCC_HSE_IsReady
	HSIPLLON	LL_RCC_RC64MPLL_Disable
		LL_RCC_RC64MPLL_Enable
	HSIPLLRDY	LL_RCC_RC64MPLL_IsReady
	HSIRDY	LL_RCC_HSI_IsReady
	LOCKDET_NSTOP	LL_RCC_LOCKDET_NSTOP_GetTime
		LL_RCC_LOCKDET_NSTOP_SetTime
	LSEBYP	LL_RCC_LSE_DisableBypass
		LL_RCC_LSE_EnableBypass
		LL_RCC_LSE_IsBypassEnabled
	LSEON	LL_RCC_LSE_Disable
		LL_RCC_LSE_Enable
		LL_RCC_LSE_IsEnabled
	LSERDY	LL_RCC_LSE_IsReady
	LSION	LL_RCC_LSI_Disable
		LL_RCC_LSI_Enable
		LL_RCC_LSI_IsEnabled
	LSIRDY	LL_RCC_LSI_IsReady
CSCMDR	CLKSYSDIV_REQ	LL_RCC_SwitchRC64MPLLPrescaler
	EOFSEQ_IE	LL_RCC_DisableIT_EOFSEQ
		LL_RCC_EnableIT_EOFSEQ
		LL_RCC_IsEnabledIT_EOFSEQ
	EOFSEQ_IRQ	LL_RCC_ClearFlag_EOFSEQ
		LL_RCC_IsActiveFlag_EOFSEQ
	REQUEST	LL_RCC_RequestSwitchRC64MPLLClock
	STATUS	LL_RCC_HSE_GetCapacitorTuning
CSR	LOCKUPRSTF	LL_RCC_IsActiveFlag_LOCKUPRST

Register	Field	Function
CSR	PADRSTF	LL_RCC_IsActiveFlag_PADRST
	PORRSTF	LL_RCC_IsActiveFlag_PORRST
	RMVF	LL_RCC_ClearResetFlags
	SFTRSTF	LL_RCC_IsActiveFlag_SFTRST
	WDGRSTF	LL_RCC_IsActiveFlag_WDGRST
CSSWCR	HSITRIM	LL_RCC_HSI_SetCalibTrimming
	LSEDRV	LL_RCC_LSE_GetDriveCapability
		LL_RCC_LSE_SetDriveCapability
	LSISWBW	LL_RCC_LSI_SetTrimming
CSSWCR/ ICSCR	LSITRIM	LL_RCC_LSI_GetTrimming
ICSCR	LSITRIMEN	LL_RCC_LSI_DisableTrimming
		LL_RCC_LSI_EnableTrimming
		LL_RCC_LSI_IsTrimmingEnabled
ICSCR/ CSSWCR	HSITRIM	LL_RCC_HSI_GetCalibTrimming
RFSWHSECR	GMC	LL_RCC_HSE_GetCurrentControl
		LL_RCC_HSE_SetCurrentControl
	SATRG	LL_RCC_HSE_GetSenseAmplifier
		LL_RCC_HSE_SetSenseAmplifier
	SWXOTUNE	LL_RCC_HSE_GetCapacitorTuning
		LL_RCC_HSE_SetCapacitorTuning

66.15 RTC

Table 39. Correspondence between RTC registers and RTC low-layer driver functions

Register	Field	Function
ALRMAR	DT	LL_RTC_ALMA_GetDay
		LL_RTC_ALMA_SetDay
	DU	LL_RTC_ALMA_GetDay
		LL_RTC_ALMA_GetWeekDay
		LL_RTC_ALMA_SetDay
		LL_RTC_ALMA_SetWeekDay
	HT	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	HU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetHour
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetHour
	MNT	LL_RTC_ALMA_ConfigTime

Register	Field	Function
ALRMAR	MNT	LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MNU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetMinute
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetMinute
	MSK1	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK2	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK3	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	MSK4	LL_RTC_ALMA_GetMask
		LL_RTC_ALMA_SetMask
	PM	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetTimeFormat
		LL_RTC_ALMA_SetTimeFormat
	ST	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	SU	LL_RTC_ALMA_ConfigTime
		LL_RTC_ALMA_GetSecond
		LL_RTC_ALMA_GetTime
		LL_RTC_ALMA_SetSecond
	WDSEL	LL_RTC_ALMA_DisableWeekday
		LL_RTC_ALMA_EnableWeekday
ALRMASSR	MASKSS	LL_RTC_ALMA_GetSubSecondMask
		LL_RTC_ALMA_SetSubSecondMask
	SS	LL_RTC_ALMA_GetSubSecond
		LL_RTC_ALMA_SetSubSecond
BKPxR	BKP	LL_RTC_BAK_GetRegister
		LL_RTC_BAK_SetRegister
CALR	CALM	LL_RTC_CAL_GetMinus
		LL_RTC_CAL_SetMinus
	CALP	LL_RTC_CAL_IsPulseInserted
		LL_RTC_CAL_SetPulse
	CALW16	LL_RTC_CAL_GetPeriod
		LL_RTC_CAL_SetPeriod
	CALW8	LL_RTC_CAL_GetPeriod

Register	Field	Function
CALR	CALW8	LL_RTC_CAL_SetPeriod
CR	ADD1H	LL_RTC_TIME_IncHour
	ALRAE	LL_RTC_ALMA_Disable
		LL_RTC_ALMA_Enable
	ALRAIE	LL_RTC_DisableIT_ALRA
		LL_RTC_EnableIT_ALRA
		LL_RTC_IsEnabledIT_ALRA
	BKP	LL_RTC_TIME_DisableDayLightStore
		LL_RTC_TIME_EnableDayLightStore
		LL_RTC_TIME_IsDayLightStoreEnabled
	BYPSHAD	LL_RTC_DisableShadowRegBypass
		LL_RTC_EnableShadowRegBypass
		LL_RTC_IsShadowRegBypassEnabled
	COE	LL_RTC_CAL_GetOutputFreq
		LL_RTC_CAL_SetOutputFreq
	COSEL	LL_RTC_CAL_GetOutputFreq
		LL_RTC_CAL_SetOutputFreq
	FMT	LL_RTC_GetHourFormat
		LL_RTC_SetHourFormat
	OSEL	LL_RTC_GetAlarmOutEvent
		LL_RTC_SetAlarmOutEvent
	POL	LL_RTC_GetOutputPolarity
		LL_RTC_SetOutputPolarity
	SUB1H	LL_RTC_TIME_DecHour
	WUCKSEL	LL_RTC_WAKEUP_GetClock
		LL_RTC_WAKEUP_SetClock
	WUTE	LL_RTC_WAKEUP_Disable
		LL_RTC_WAKEUP_Enable
		LL_RTC_WAKEUP_IsEnabled
	WUTIE	LL_RTC_DisableIT_WUT
		LL_RTC_EnableIT_WUT
		LL_RTC_IsEnabledIT_WUT
DR	DT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	DU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetDay
		LL_RTC_DATE_SetDay
	MT	LL_RTC_DATE_Config

Register	Field	Function
DR	MT	LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	MU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetMonth
		LL_RTC_DATE_SetMonth
	WDU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetWeekDay
		LL_RTC_DATE_SetWeekDay
	YT	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
	YU	LL_RTC_DATE_Config
		LL_RTC_DATE_Get
		LL_RTC_DATE_GetYear
		LL_RTC_DATE_SetYear
ISR	ALRAF	LL_RTC_ClearFlag_ALRA
		LL_RTC_IsActiveFlag_ALRA
	ALRAWF	LL_RTC_IsActiveFlag_ALRAW
	INIT	LL_RTC_DisableInitMode
		LL_RTC_EnableInitMode
	INITF	LL_RTC_IsActiveFlag_INIT
	INITS	LL_RTC_IsActiveFlag_INITS
	RECALPF	LL_RTC_IsActiveFlag_RECALP
	RSF	LL_RTC_ClearFlag_RS
		LL_RTC_IsActiveFlag_RS
	SHPF	LL_RTC_IsActiveFlag_SHP
	WUTF	LL_RTC_ClearFlag_WUT
		LL_RTC_IsActiveFlag_WUT
	WUTWF	LL_RTC_IsActiveFlag_WUTW
PRER	PREDIV_A	LL_RTC_GetAsynchPrescaler
		LL_RTC_SetAsynchPrescaler
	PREDIV_S	LL_RTC_GetSynchPrescaler
		LL_RTC_SetSynchPrescaler
SHIFTR	ADD1S	LL_RTC_TIME_Synchronize
	SUBFS	LL_RTC_TIME_Synchronize
SSR	SS	LL_RTC_TIME_GetSubSecond
TR	HT	LL_RTC_TIME_Config

Register	Field	Function
TR	HT	LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	HU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetHour
		LL_RTC_TIME_SetHour
	MNT	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	MNU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetMinute
		LL_RTC_TIME_SetMinute
	PM	LL_RTC_TIME_Config
		LL_RTC_TIME_GetFormat
		LL_RTC_TIME_SetFormat
	ST	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
	SU	LL_RTC_TIME_Config
		LL_RTC_TIME_Get
		LL_RTC_TIME_GetSecond
		LL_RTC_TIME_SetSecond
WPR	KEY	LL_RTC_DisableWriteProtection
		LL_RTC_EnableWriteProtection
WUTR	WUT	LL_RTC_WAKEUP_GetAutoReload
		LL_RTC_WAKEUP_SetAutoReload

66.16 SPI

Table 40. **Correspondence between SPI registers and SPI low-layer driver functions**

Register	Field	Function
CR1	BIDIMODE	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	BIDIOE	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	BR	LL_SPI_GetBaudRatePrescaler
		LL_SPI_SetBaudRatePrescaler

Register	Field	Function
CR1	CPHA	LL_SPI_GetClockPhase
		LL_SPI_SetClockPhase
	CPOL	LL_SPI_GetClockPolarity
		LL_SPI_SetClockPolarity
	CRCEN	LL_SPI_DisableCRC
		LL_SPI_EnableCRC
		LL_SPI_IsEnabledCRC
	CRCL	LL_SPI_GetCRCWidth
		LL_SPI_SetCRCWidth
	CRCNEXT	LL_SPI_SetCRCNext
	LSBFIRST	LL_SPI_GetTransferBitOrder
		LL_SPI_SetTransferBitOrder
	MSTR	LL_SPI_GetMode
		LL_SPI_SetMode
	RXONLY	LL_SPI_GetTransferDirection
		LL_SPI_SetTransferDirection
	SPE	LL_SPI_Disable
		LL_SPI_Enable
		LL_SPI_IsEnabled
	SSI	LL_SPI_GetMode
		LL_SPI_SetMode
	SSM	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
CR2	DS	LL_SPI_GetDataWidth
		LL_SPI_SetDataWidth
	ERRIE	LL_SPI_DisableIT_ERR
		LL_SPI_EnableIT_ERR
		LL_SPI_IsEnabledIT_ERR
	FRF	LL_SPI_GetStandard
		LL_SPI_SetStandard
	FRXTH	LL_SPI_GetRxFIFOThreshold
		LL_SPI_SetRxFIFOThreshold
	LDMARX	LL_SPI_GetDMAParity_RX
		LL_SPI_SetDMAParity_RX
	LDMATX	LL_SPI_GetDMAParity_TX
		LL_SPI_SetDMAParity_TX
	NSSP	LL_SPI_DisableNSSPulseMgt
		LL_SPI_EnableNSSPulseMgt
		LL_SPI_IsEnabledNSSPulse
	RXDMAEN	LL_SPI_DisableDMAReq_RX
		LL_SPI_EnableDMAReq_RX

Register	Field	Function
CR2	RXDMAEN	LL_SPI_IsEnabledDMAReq_RX
	RXNEIE	LL_SPI_DisableIT_RXNE
		LL_SPI_EnableIT_RXNE
		LL_SPI_IsEnabledIT_RXNE
	SSOE	LL_SPI_GetNSSMode
		LL_SPI_SetNSSMode
	TXDMAEN	LL_SPI_DisableDMAReq_TX
		LL_SPI_EnableDMAReq_TX
		LL_SPI_IsEnabledDMAReq_TX
	TXEIE	LL_SPI_DisableIT_TXE
		LL_SPI_EnableIT_TXE
		LL_SPI_IsEnabledIT_TXE
CRCPR	CRCPOLY	LL_SPI_GetCRCPolynomial
		LL_SPI_SetCRCPolynomial
DR	DR	LL_SPI_DMA_GetRegAddr
		LL_SPI_ReceiveData16
		LL_SPI_ReceiveData8
		LL_SPI_TransmitData16
		LL_SPI_TransmitData8
RXCRCR	RXCRC	LL_SPI_GetRxCRC
SR	BSY	LL_SPI_IsActiveFlag_BSY
	CRCERR	LL_SPI_ClearFlag_CRCERR
		LL_SPI_IsActiveFlag_CRCERR
	FRE	LL_SPI_ClearFlag_FRE
		LL_SPI_IsActiveFlag_FRE
	FRLVL	LL_SPI_GetRxFIFOLevel
	FTLVL	LL_SPI_GetTxFIFOLevel
	MODF	LL_SPI_ClearFlag_MODF
		LL_SPI_IsActiveFlag_MODF
	OVR	LL_SPI_ClearFlag_OVR
		LL_SPI_IsActiveFlag_OVR
	RXNE	LL_SPI_IsActiveFlag_RXNE
	TXE	LL_SPI_IsActiveFlag_TXE
TXCRCR	TXCRC	LL_SPI_GetTxCRC

66.17 SYSTEM

Table 41. **Correspondence between SYSTEM registers and SYSTEM low-layer driver functions**

Register	Field	Function
FLASH_CONFIG	WAIT_STATE	LL_FLASH_GetLatency
		LL_FLASH_SetLatency

66.18 TIM

Table 42. Correspondence between TIM registers and TIM low-layer driver functions

Register	Field	Function
AF1	BKINE	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BKINP	LL_TIM_SetBreakInputSourcePolarity
AF2	BK2INE	LL_TIM_DisableBreakInputSource
		LL_TIM_EnableBreakInputSource
	BK2INP	LL_TIM_SetBreakInputSourcePolarity
ARR	ARR	LL_TIM_GetAutoReload
		LL_TIM_SetAutoReload
BDTR	AOE	LL_TIM_DisableAutomaticOutput
		LL_TIM_EnableAutomaticOutput
		LL_TIM_IsEnabledAutomaticOutput
	BKBID	LL_TIM_ConfigBRK
	BKDSRM	LL_TIM_DisarmBRK
		LL_TIM_ReArmBRK
	BKE	LL_TIM_DisableBRK
		LL_TIM_EnableBRK
	BKF	LL_TIM_ConfigBRK
	BKP	LL_TIM_ConfigBRK
	DTG	LL_TIM_OC_SetDeadTime
	LOCK	LL_TIM_CC_SetLockLevel
	MOE	LL_TIM_DisableAllOutputs
		LL_TIM_EnableAllOutputs
		LL_TIM_IsEnabledAllOutputs
	OSSI	LL_TIM_SetOffStates
	OSSR	LL_TIM_SetOffStates
CCER	CC1E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC1NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC1P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity

Register	Field	Function
CCER	CC1P	LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC2NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC2P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NE	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC3NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC3P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC4E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel

Register	Field	Function
CCER	CC4NP	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
	CC4P	LL_TIM_IC_Config
		LL_TIM_IC_GetPolarity
		LL_TIM_IC_SetPolarity
		LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC5E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC5P	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
	CC6E	LL_TIM_CC_DisableChannel
		LL_TIM_CC_EnableChannel
		LL_TIM_CC_IsEnabledChannel
	CC6P	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetPolarity
		LL_TIM_OC_SetPolarity
CCMR1	CC1S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC2S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC1F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC1PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC2F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC2PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler

Register	Field	Function
CCMR1	OC1CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC1FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC1M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC1PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC2CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC2FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC2M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC2PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR2	CC3S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	CC4S	LL_TIM_IC_Config
		LL_TIM_IC_GetActiveInput
		LL_TIM_IC_SetActiveInput
		LL_TIM_OC_ConfigOutput
	IC3F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC3PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler
		LL_TIM_IC_SetPrescaler
	IC4F	LL_TIM_IC_Config
		LL_TIM_IC_GetFilter
		LL_TIM_IC_SetFilter
	IC4PSC	LL_TIM_IC_Config
		LL_TIM_IC_GetPrescaler

Register	Field	Function
CCMR2	IC4PSC	LL_TIM_IC_SetPrescaler
	OC3CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC3FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC3M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC3PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC4CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC4FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC4M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC4PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCMR3	CC5S	LL_TIM_OC_ConfigOutput
	CC6S	LL_TIM_OC_ConfigOutput
	OC5CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC5FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast
		LL_TIM_OC_IsEnabledFast
	OC5M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC5PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
	OC6CE	LL_TIM_OC_DisableClear
		LL_TIM_OC_EnableClear
		LL_TIM_OC_IsEnabledClear
	OC6FE	LL_TIM_OC_DisableFast
		LL_TIM_OC_EnableFast

Register	Field	Function
CCMR3	OC6FE	LL_TIM_OC_IsEnabledFast
	OC6M	LL_TIM_OC_GetMode
		LL_TIM_OC_SetMode
	OC6PE	LL_TIM_OC_DisablePreload
		LL_TIM_OC_EnablePreload
		LL_TIM_OC_IsEnabledPreload
CCR1	CCR1	LL_TIM_IC_GetCaptureCH1
		LL_TIM_OC_GetCompareCH1
		LL_TIM_OC_SetCompareCH1
CCR2	CCR2	LL_TIM_IC_GetCaptureCH2
		LL_TIM_OC_GetCompareCH2
		LL_TIM_OC_SetCompareCH2
CCR3	CCR3	LL_TIM_IC_GetCaptureCH3
		LL_TIM_OC_GetCompareCH3
		LL_TIM_OC_SetCompareCH3
CCR4	CCR4	LL_TIM_IC_GetCaptureCH4
		LL_TIM_OC_GetCompareCH4
		LL_TIM_OC_SetCompareCH4
CNT	CNT	LL_TIM_GetCounter
		LL_TIM_SetCounter
CR1	ARPE	LL_TIM_DisableARRPreload
		LL_TIM_EnableARRPreload
		LL_TIM_IsEnabledARRPreload
	CEN	LL_TIM_DisableCounter
		LL_TIM_EnableCounter
		LL_TIM_IsEnabledCounter
	CKD	LL_TIM_GetClockDivision
		LL_TIM_SetClockDivision
	CMS	LL_TIM_GetCounterMode
		LL_TIM_SetCounterMode
	DIR	LL_TIM_GetCounterMode
		LL_TIM_GetDirection
		LL_TIM_SetCounterMode
	OPM	LL_TIM_GetOnePulseMode
		LL_TIM_SetOnePulseMode
	UDIS	LL_TIM_DisableUpdateEvent
		LL_TIM_EnableUpdateEvent
		LL_TIM_IsEnabledUpdateEvent
	UIFREMAP	LL_TIM_DisableUIFRemap
		LL_TIM_EnableUIFRemap
	URS	LL_TIM_GetUpdateSource

Register	Field	Function
CR1	URS	LL_TIM_SetUpdateSource
CR2	CCDS	LL_TIM_CC_GetDMAReqTrigger
		LL_TIM_CC_SetDMAReqTrigger
	CCPC	LL_TIM_CC_DisablePreload
		LL_TIM_CC_EnablePreload
	CCUS	LL_TIM_CC_SetUpdate
	OIS1	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS2N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS3N	LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS4	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS5	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	OIS6	LL_TIM_OC_ConfigOutput
		LL_TIM_OC_GetIdleState
		LL_TIM_OC_SetIdleState
	TI1S	LL_TIM_IC_DisableXORCombination
		LL_TIM_IC_EnableXORCombination
		LL_TIM_IC_IsEnabledXORCombination
DCR	DBA	LL_TIM_ConfigDMABurst
	DBL	LL_TIM_ConfigDMABurst
DIER	BIE	LL_TIM_DisableIT_BRK
		LL_TIM_EnableIT_BRK
		LL_TIM_IsEnabledIT_BRK
	CC1DE	LL_TIM_DisableDMAReq_CC1
		LL_TIM_EnableDMAReq_CC1
		LL_TIM_IsEnabledDMAReq_CC1
	CC1IE	LL_TIM_DisableIT_CC1
		LL_TIM_EnableIT_CC1

Register	Field	Function
DIER	CC1IE	LL_TIM_IsEnabledIT_CC1
	CC2DE	LL_TIM_DisableDMAReq_CC2
		LL_TIM_EnableDMAReq_CC2
		LL_TIM_IsEnabledDMAReq_CC2
	CC2IE	LL_TIM_DisableIT_CC2
		LL_TIM_EnableIT_CC2
		LL_TIM_IsEnabledIT_CC2
	CC3DE	LL_TIM_DisableDMAReq_CC3
		LL_TIM_EnableDMAReq_CC3
		LL_TIM_IsEnabledDMAReq_CC3
	CC3IE	LL_TIM_DisableIT_CC3
		LL_TIM_EnableIT_CC3
		LL_TIM_IsEnabledIT_CC3
	CC4DE	LL_TIM_DisableDMAReq_CC4
		LL_TIM_EnableDMAReq_CC4
		LL_TIM_IsEnabledDMAReq_CC4
	CC4IE	LL_TIM_DisableIT_CC4
		LL_TIM_EnableIT_CC4
		LL_TIM_IsEnabledIT_CC4
	COMIE	LL_TIM_DisableIT_COM
		LL_TIM_EnableIT_COM
		LL_TIM_IsEnabledIT_COM
	TIE	LL_TIM_DisableIT_TRIG
		LL_TIM_EnableIT_TRIG
		LL_TIM_IsEnabledIT_TRIG
	UDE	LL_TIM_DisableDMAReq_UPDATE
		LL_TIM_EnableDMAReq_UPDATE
		LL_TIM_IsEnabledDMAReq_UPDATE
	UIE	LL_TIM_DisableIT_UPDATE
		LL_TIM_EnableIT_UPDATE
		LL_TIM_IsEnabledIT_UPDATE
EGR	BG	LL_TIM_GenerateEvent_BRK
	CC1G	LL_TIM_GenerateEvent_CC1
	CC2G	LL_TIM_GenerateEvent_CC2
	CC3G	LL_TIM_GenerateEvent_CC3
	CC4G	LL_TIM_GenerateEvent_CC4
	COMG	LL_TIM_GenerateEvent_COM
	TG	LL_TIM_GenerateEvent_TRIG
	UG	LL_TIM_GenerateEvent_UPDATE
PSC	PSC	LL_TIM_GetPrescaler
		LL_TIM_SetPrescaler

Register	Field	Function
RCR	REP	LL_TIM_GetRepetitionCounter
		LL_TIM_SetRepetitionCounter
SMCR	ECE	LL_TIM_DisableExternalClock
		LL_TIM_EnableExternalClock
		LL_TIM_IsEnabledExternalClock
		LL_TIM_SetClockSource
	ETF	LL_TIM_ConfigETR
	ETP	LL_TIM_ConfigETR
	ETPS	LL_TIM_ConfigETR
	OCCS	LL_TIM_SetOCRefClearInputSource
	SMS	LL_TIM_SetClockSource
		LL_TIM_SetEncoderMode
		LL_TIM_SetSlaveMode
	TS	LL_TIM_SetTriggerInput
SR	BIF	LL_TIM_ClearFlag_BRK
		LL_TIM_IsActiveFlag_BRK
	CC1IF	LL_TIM_ClearFlag_CC1
		LL_TIM_IsActiveFlag_CC1
	CC1OF	LL_TIM_ClearFlag_CC1OVR
		LL_TIM_IsActiveFlag_CC1OVR
	CC2IF	LL_TIM_ClearFlag_CC2
		LL_TIM_IsActiveFlag_CC2
	CC2OF	LL_TIM_ClearFlag_CC2OVR
		LL_TIM_IsActiveFlag_CC2OVR
	CC3IF	LL_TIM_ClearFlag_CC3
		LL_TIM_IsActiveFlag_CC3
	CC3OF	LL_TIM_ClearFlag_CC3OVR
		LL_TIM_IsActiveFlag_CC3OVR
	CC4IF	LL_TIM_ClearFlag_CC4
		LL_TIM_IsActiveFlag_CC4
	CC4OF	LL_TIM_ClearFlag_CC4OVR
		LL_TIM_IsActiveFlag_CC4OVR
	COMIF	LL_TIM_ClearFlag_COM
		LL_TIM_IsActiveFlag_COM
	TIF	LL_TIM_ClearFlag_TRIG
		LL_TIM_IsActiveFlag_TRIG
	UIF	LL_TIM_ClearFlag_UPDATE
		LL_TIM_IsActiveFlag_UPDATE
TIM17_OR1	TI1_RMP	LL_TIM_SetRemap

66.19 USART

Table 43. Correspondence between USART registers and USART low-layer driver functions

Register	Field	Function
BRR	BRR	LL_USART_GetBaudRate
		LL_USART_SetBaudRate
CR1	CMIE	LL_USART_DisableIT_CM
		LL_USART_EnableIT_CM
		LL_USART_IsEnabledIT_CM
	DEAT	LL_USART_GetDEAssertionTime
		LL_USART_SetDEAssertionTime
	DEDT	LL_USART_GetDEDeassertionTime
		LL_USART_SetDEDeassertionTime
	EOBIE	LL_USART_DisableIT_EOB
		LL_USART_EnableIT_EOB
		LL_USART_IsEnabledIT_EOB
	FIFOEN	LL_USART_DisableFIFO
		LL_USART_EnableFIFO
		LL_USART_IsEnabledFIFO
	IDLEIE	LL_USART_DisableIT_IDLE
		LL_USART_EnableIT_IDLE
		LL_USART_IsEnabledIT_IDLE
	M0	LL_USART_ConfigCharacter
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	M1	LL_USART_ConfigCharacter
		LL_USART_GetDataWidth
		LL_USART_SetDataWidth
	MME	LL_USART_DisableMuteMode
		LL_USART_EnableMuteMode
		LL_USART_IsEnabledMuteMode
	OVER8	LL_USART_GetOverSampling
		LL_USART_SetOverSampling
	PCE	LL_USART_ConfigCharacter
		LL_USART_GetParity
		LL_USART_SetParity
	PEIE	LL_USART_DisableIT_PE
		LL_USART_EnableIT_PE
		LL_USART_IsEnabledIT_PE
	PS	LL_USART_ConfigCharacter
		LL_USART_GetParity
		LL_USART_SetParity

Register	Field	Function
CR1	RE	LL_USART_DisableDirectionRx
		LL_USART_EnableDirectionRx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	RTOIE	LL_USART_DisableIT_RTO
		LL_USART_EnableIT_RTO
		LL_USART_IsEnabledIT_RTO
	RXFFIE	LL_USART_DisableIT_RXFF
		LL_USART_EnableIT_RXFF
		LL_USART_IsEnabledIT_RXFF
	RXNEIE_RXFNEIE	LL_USART_DisableIT_RXNE_RXFNE
		LL_USART_EnableIT_RXNE_RXFNE
		LL_USART_IsEnabledIT_RXNE_RXFNE
	TCIE	LL_USART_DisableIT_TC
		LL_USART_EnableIT_TC
		LL_USART_IsEnabledIT_TC
	TE	LL_USART_DisableDirectionTx
		LL_USART_EnableDirectionTx
		LL_USART_GetTransferDirection
		LL_USART_SetTransferDirection
	TXEIE_TXFNFIE	LL_USART_DisableIT_TXE_TXFNF
		LL_USART_EnableIT_TXE_TXFNF
		LL_USART_IsEnabledIT_TXE_TXFNF
	TXFEIE	LL_USART_DisableIT_TXFE
		LL_USART_EnableIT_TXFE
		LL_USART_IsEnabledIT_TXFE
	UE	LL_USART_Disable
		LL_USART_Enable
		LL_USART_IsEnabled
	UESM	LL_USART_DisableInStopMode
		LL_USART_EnableInStopMode
		LL_USART_IsEnabledInStopMode
	WAKE	LL_USART_GetWakeUpMethod
		LL_USART_SetWakeUpMethod
CR2	ABREN	LL_USART_DisableAutoBaudRate
		LL_USART_EnableAutoBaudRate
		LL_USART_IsEnabledAutoBaud
	ABRMODE	LL_USART_GetAutoBaudRateMode
		LL_USART_SetAutoBaudRateMode
	ADD	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddress

Register	Field	Function
CR2	ADDM7	LL_USART_ConfigNodeAddress
		LL_USART_GetNodeAddressLen
	CLKEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSCLKOutput
		LL_USART_EnableSCLKOutput
		LL_USART_IsEnabledSCLKOutput
	CPHA	LL_USART_ConfigClock
		LL_USART_GetClockPhase
		LL_USART_SetClockPhase
	CPOL	LL_USART_ConfigClock
		LL_USART_GetClockPolarity
		LL_USART_SetClockPolarity
	DATAINV	LL_USART_GetBinaryDataLogic
		LL_USART_SetBinaryDataLogic
	DIS_NSS	LL_USART_DisableSPISlaveSelect
		LL_USART_EnableSPISlaveSelect
		LL_USART_IsEnabledSPISlaveSelect
	LBCL	LL_USART_ConfigClock
		LL_USART_GetLastClkPulseOutput
		LL_USART_SetLastClkPulseOutput
	LBDIE	LL_USART_DisableIT_LBD
		LL_USART_EnableIT_LBD
		LL_USART_IsEnabledIT_LBD
	LBDL	LL_USART_GetLINBrkDetectionLen
		LL_USART_SetLINBrkDetectionLen
	LINEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableLIN
		LL_USART_EnableLIN
		LL_USART_IsEnabledLIN

Register	Field	Function
CR2	MSBFIRST	LL_USART_GetTransferBitOrder
		LL_USART_SetTransferBitOrder
	RTOEN	LL_USART_DisableRxTimeout
		LL_USART_EnableRxTimeout
		LL_USART_IsEnabledRxTimeout
	RXINV	LL_USART_GetRXPinLevel
		LL_USART_SetRXPinLevel
	SLVEN	LL_USART_DisableSPISlave
		LL_USART_EnableSPISlave
		LL_USART_IsEnabledSPISlave
	STOP	LL_USART_ConfigCharacter
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigSmartcardMode
		LL_USART_GetStopBitsLength
		LL_USART_SetStopBitsLength
	SWAP	LL_USART_GetTXRXSwap
		LL_USART_SetTXRXSwap
	TXINV	LL_USART_GetTXPinLevel
		LL_USART_SetTXPinLevel
CR3	CTSE	LL_USART_DisableCTSHWFlowCtrl
		LL_USART_EnableCTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	CTSIE	LL_USART_DisableIT_CTS
		LL_USART_EnableIT_CTS
		LL_USART_IsEnabledIT_CTS
	DDRE	LL_USART_DisableDMADeactOnRxErr
		LL_USART_EnableDMADeactOnRxErr
		LL_USART_IsEnabledDMADeactOnRxErr
	DEM	LL_USART_DisableDEMode
		LL_USART_EnableDEMode
		LL_USART_IsEnabledDEMode
	DEP	LL_USART_GetDESignalPolarity
		LL_USART_SetDESignalPolarity
	DMAR	LL_USART_DisableDMAReq_RX
		LL_USART_EnableDMAReq_RX
		LL_USART_IsEnabledDMAReq_RX
	DMAT	LL_USART_DisableDMAReq_TX
		LL_USART_EnableDMAReq_TX
		LL_USART_IsEnabledDMAReq_TX

Register	Field	Function
CR3	EIE	LL_USART_DisableIT_ERROR
		LL_USART_EnableIT_ERROR
		LL_USART_IsEnabledIT_ERROR
	HDSEL	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableHalfDuplex
		LL_USART_EnableHalfDuplex
		LL_USART_IsEnabledHalfDuplex
	IREN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableIrda
		LL_USART_EnableIrda
		LL_USART_IsEnabledIrda
	IRLP	LL_USART_GetIrdaPowerMode
		LL_USART_SetIrdaPowerMode
	NACK	LL_USART_DisableSmartcardNACK
		LL_USART_EnableSmartcardNACK
		LL_USART_IsEnabledSmartcardNACK
	ONEBIT	LL_USART_DisableOneBitSamp
		LL_USART_EnableOneBitSamp
		LL_USART_IsEnabledOneBitSamp
	OVRDIS	LL_USART_DisableOverrunDetect
		LL_USART_EnableOverrunDetect
		LL_USART_IsEnabledOverrunDetect
	RTSE	LL_USART_DisableRTSHWFlowCtrl
		LL_USART_EnableRTSHWFlowCtrl
		LL_USART_GetHWFlowCtrl
		LL_USART_SetHWFlowCtrl
	RXFTCFG	LL_USART_ConfigFIFOsThreshold
		LL_USART_GetRXFIFOThreshold
		LL_USART_SetRXFIFOThreshold
	RXFTIE	LL_USART_DisableIT_RXFT

Register	Field	Function
CR3	RXFTIE	LL_USART_EnableIT_RXFT
		LL_USART_IsEnabledIT_RXFT
	SCARCNT	LL_USART_GetSmartcardAutoRetryCount
		LL_USART_SetSmartcardAutoRetryCount
	SCEN	LL_USART_ConfigAsyncMode
		LL_USART_ConfigHalfDuplexMode
		LL_USART_ConfigIrdaMode
		LL_USART_ConfigLINMode
		LL_USART_ConfigMultiProcessMode
		LL_USART_ConfigSmartcardMode
		LL_USART_ConfigSyncMode
		LL_USART_DisableSmartcard
		LL_USART_EnableSmartcard
		LL_USART_IsEnabledSmartcard
	TCBGTIE	LL_USART_DisableIT_TCBGT
		LL_USART_EnableIT_TCBGT
		LL_USART_IsEnabledIT_TCBGT
	TXFTCFG	LL_USART_ConfigFIFOsThreshold
		LL_USART_GetTXFIFOThreshold
		LL_USART_SetTXFIFOThreshold
	TXFTIE	LL_USART_DisableIT_TXFT
		LL_USART_EnableIT_TXFT
		LL_USART_IsEnabledIT_TXFT
	WUFIE	LL_USART_DisableIT_WKUP
		LL_USART_EnableIT_WKUP
		LL_USART_IsEnabledIT_WKUP
	WUS	LL_USART_GetWKUPType
		LL_USART_SetWKUPType
GTPR	GT	LL_USART_GetSmartcardGuardTime
		LL_USART_SetSmartcardGuardTime
	PSC	LL_USART_GetIrdaPrescaler
		LL_USART_GetSmartcardPrescaler
		LL_USART_SetIrdaPrescaler
		LL_USART_SetSmartcardPrescaler
ICR	CMCF	LL_USART_ClearFlag_CM
	CTSCF	LL_USART_ClearFlag_nCTS
	EOBCF	LL_USART_ClearFlag_EOB
	FECF	LL_USART_ClearFlag_FE
	IDLECF	LL_USART_ClearFlag_IDLE
	LBDCF	LL_USART_ClearFlag_LBD
	NECF	LL_USART_ClearFlag_NE

Register	Field	Function
ICR	ORECF	LL_USART_ClearFlag_ORE
	PECF	LL_USART_ClearFlag_PE
	RTOCF	LL_USART_ClearFlag_RTO
	TCBGTCF	LL_USART_ClearFlag_TCBGT
	TCCF	LL_USART_ClearFlag_TC
	TXFECF	LL_USART_ClearFlag_TXFE
	UDRCF	LL_USART_ClearFlag_UDR
	WUCF	LL_USART_ClearFlag_WKUP
ISR	ABRE	LL_USART_IsActiveFlag_ABRE
	ABRF	LL_USART_IsActiveFlag_ABR
	BUSY	LL_USART_IsActiveFlag_BUSY
	CMF	LL_USART_IsActiveFlag_CM
	CTS	LL_USART_IsActiveFlag_CTS
	CTSIF	LL_USART_IsActiveFlag_nCTS
	EOBF	LL_USART_IsActiveFlag_EOB
	FE	LL_USART_IsActiveFlag_FE
	IDLE	LL_USART_IsActiveFlag_IDLE
	LBDF	LL_USART_IsActiveFlag_LBD
	NE	LL_USART_IsActiveFlag_NE
	ORE	LL_USART_IsActiveFlag_ORE
	PE	LL_USART_IsActiveFlag_PE
	REACK	LL_USART_IsActiveFlag_REACK
	RTOF	LL_USART_IsActiveFlag_RTO
	RWU	LL_USART_IsActiveFlag_RWU
	RXFF	LL_USART_IsActiveFlag_RXFF
	RXFT	LL_USART_IsActiveFlag_RXFT
	RXNE_RXFNE	LL_USART_IsActiveFlag_RXNE_RXFNE
	SBKF	LL_USART_IsActiveFlag_SBK
	TC	LL_USART_IsActiveFlag_TC
	TCBGT	LL_USART_IsActiveFlag_TCBGT
	TEACK	LL_USART_IsActiveFlag_TEACK
	TXE_TXFNF	LL_USART_IsActiveFlag_TXE_TXFNF
	TXFE	LL_USART_IsActiveFlag_TXFE
	TXFT	LL_USART_IsActiveFlag_TXFT
	UDR	LL_USART_IsActiveFlag_UDR
	WUF	LL_USART_IsActiveFlag_WKUP
PRESC	PRESCALER	LL_USART_GetPrescaler
		LL_USART_SetPrescaler
RDR	RDR	LL_USART_DMA_GetRegAddr
		LL_USART_ReceiveData8
		LL_USART_ReceiveData9

Register	Field	Function
RQR	ABRRQ	LL_USART_RequestAutoBaudRate
	MMRQ	LL_USART_RequestEnterMuteMode
	RXFRQ	LL_USART_RequestRxDataFlush
	SBKRQ	LL_USART_RequestBreakSending
	TXFRQ	LL_USART_RequestTxDataFlush
RTOR	BLEN	LL_USART_GetBlockLength
		LL_USART_SetBlockLength
	RTO	LL_USART_GetRxTimeout
		LL_USART_SetRxTimeout
TDR	TDR	LL_USART_DMA_GetRegAddr
		LL_USART_TransmitData8
		LL_USART_TransmitData9

67 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 STM32WB0 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: stm32wb0xx_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 (stm32wb0xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32wb0xx_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 (stm32wb0xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32wb0xx_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_stm32wb0xx.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 stm32wb0xx_hal_msp.c. A template is provided in the HAL driver folders (stm32wb0xx_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 stm32wb0xx_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 stm32wb0xx_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 stm32wb0xx_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 stm32wb0xx_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 44. Document revision history

Date	Revision	Changes
14-Jun-2024	1	Initial release.
24-Sep-2025	2	Updated title and Introduction on cover page. Removed from the "STM32WB0 low-power management features" item in Section 3.11.4: PWR the APIs: • HAL_PWREx_EnterSTOP0Mode() • HAL_PWREx_EnterSTOP1Mode() • HAL_PWREx_EnterSTOP2Mode() Added in Section 23.2.2: Peripheral Control functions the link to the API HAL_PWREx_EnterSHUTDOWNMode(). Added in Section 23.2.3: Detailed description of functions the API HAL_PWREx_EnterSHUTDOWNMode(). Minor text changes.

Contents

List of tables

List of figures